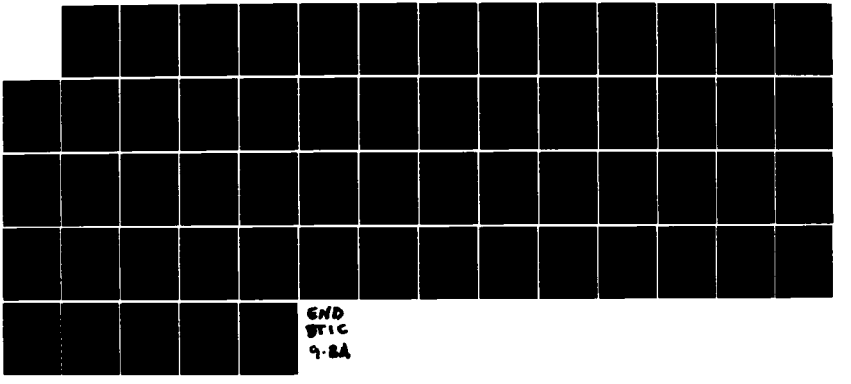
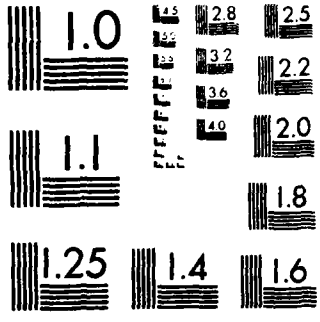
AD-A143 765 THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO (ARRAY ORIENTED PROCESSOR) ARRAY PROCESSOR(U) ELECTRONICS RESEARCH LAB ADELAIDE (AUSTRALIA) 1/1

UNCLASSIFIED D J HEILBRONN AUG 83 ERL-0286-TM F/G 9/2 NL





(12)

ERL-0286 TM

AR-002-802

AD-A143 765

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

ELECTRONICS RESEARCH LABORATORY

DEFENCE RESEARCH CENTRE SALISBURY
SOUTH AUSTRALIA

TECHNICAL MEMORANDUM

ERL-0286-TM

THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO ARRAY PROCESSOR

D.J. HEILBRONN

Technical Memoranda are of a tentative nature, representing the views of the author(s), and do not necessarily carry the authority of the Laboratory.

Approved for Public Release

COPY No.

AUGUST 1983

AR-002-802

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

ELECTRONICS RESEARCH LABORATORY

TECHNICAL MEMORANDUM

ERL-0286-TM

THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO ARRAY PROCESSOR

D.J. Heilbronn

S U M M A R Y

A high speed array processor (ARO) has been designed for the processing of radar data in real time. The operation of the ARO is supervised by a bit-slice microprocessor (the Control Processor). This document contains a description of the microcode which was written for the Control Processor to enable it to interpret PDP-11 assembler code.

POSTAL ADDRESS: Director, Electronics Research Laboratory,
Box 2151, GPO, Adelaide, South Australia, 5001.

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The Control Processor is one element of the Array Oriented Processor (ARO) which has been designed within ERL. An overview of the operation of the ARO is provided elsewhere(ref.9). The ARO works in conjunction with a host PDP 11/34 to provide high speed processing. The host delegates processing to the ARO by transferring an operating system to it, scheduling tasks and initiating it to run autonomously. The ARO has its own inbuilt processor called the Control Processor (CP) which basically serves to run the operating system and control the operation of the remainder of the ARO.

The design philosophy of the ARO is described elsewhere(ref.3,9). This document basically serves to provide sufficient detail of the microcode written for the CP to allow the microcode to be maintained.

The PDP-11 assembler language was chosen as the target language for the CP to be compatible with the host processor. The design philosophy of the CP by necessity took into account the nature of the PDP-11 instruction set. The ground rules followed in implementing the microcode were established by Mr J. Zuikelis in reference 3. A full description of the hardware of the CP is provided in reference 1. The CP is based on the Am2900 family of bit-slice microprocessor products. The principles involved in designing and microprogramming with the Am2900 family are explained in reference 7.

The CP emulates a PDP-11/34 by reading object code which is stored in the CP main memory into an instruction port (see figure 1). This PDP-11 instruction is fed to a mapping PROM which yields a microcode starting address within the microcode store. The CP then executes that microcode routine which manipulates the CP main memory in a manner identical to a PDP-11/34.

The microcode was generated using the Signetics Micro-assembler(ref.2) which is available on the IBM 370 computer which is installed at DRCS. The micro-assembler enables the user to define the micro-instruction word format and the mnemonics to suit the application. The source code (input to the micro-assembler) consists of three distinct sections - the micro instruction format section, the microp definition section (similar to macro definitions) and the program section. In order to fully appreciate the contents of this document, the reader will have to be familiar with the hardware description of the Control Processor(ref.1,3), the micro-assembler description(ref.2) and the PDP-11 instruction set(ref.8). Within these constraints, Section 5 is provided as a guide to enable the user to modify the microcode. It is anticipated that modification will only be required in the program section with the syntax and micro-instruction format sections remaining unchanged.

The complete listing of the micro-assembler output is shown in Appendices I, II and III. Instructions for using the IBM 370/3033 to generate the microcode are given in Appendix IV.

2. SAMPLE OF MICRO-ASSEMBLER OUTPUT

A sample of the micro-assembler output with the 64 bit microcode interspersed with the source input is shown in figure 2. This section depicts the fetch cycle which reads the PDP-11 code into the instruction port and the single operand PDP-11 instructions CLR, COM, INC, DEC, NEG, ADC and SBC. Note that the start address in the microcode for the fetch cycle is 31 (octal) and that the start address in the microcode of CLR1 is 374 (octal). The mnemonic NEXT causes the CP to jump to the start of the fetch cycle.

Figure 2 may be used to illustrate the operation of the CP as follows. On initialisation, the micro-program counter is forced by the host to the start of the fetch cycle of the microcode. Assume that the macro-program counter (equivalent to the PDP-11 register R7) is pointing to a location in CP main memory containing the machine code for:

```
CLR     R0      ; PDP-11 code for clear reg 0
```

The execution of the fetch cycle would cause the micro-instructions 31, 32, 33 and 374 to be invoked, followed by another fetch cycle. The macro-program counter would be updated to point to the next PDP-11 instruction in CP main memory.

3. THE MICRO-INSTRUCTION FORMAT

The micro-instruction format is identical to that described in reference 2. The fields (see Table 1) are defined as SRC REG, DST REG, MAR, ALU CONTROL, 2904 STATUS & SHIFT, MICROSEQUENCE CONTROL, SPECIAL OPERATIONS FIELD, IMMEDIATE FIELD and an ERROR FIELD to record assembly time errors. The fields are further divided into sub-fields to link to the nomenclature of references 1, 3, & 4.

TABLE 1. MICRO-INSTRUCTION FORMAT

FIELD	SUB-FIELD	WIDTH	COMMENT	DEFAULT
SRC	AMODE AFIELD	2 4	A-MODE SELECT) SOURCE REGISTER A-SOURCE SELECT)	0 0
DST	BMODE BFIELD	2 4	B-MODE SELECT) DESTINATION REG. B-SOURCE SELECT)	1 7
MARS		1	MEMORY ADDR. REG. SELECT	0
C2903	I1_4 I0 I5_8	4 1 4	ALU OPERATION SELECT ALU OPERATION SELECT ALU DEST CONTROL, SPECIAL FUNC	4(MOV_D) 1 0CH(TST)
C2904	I11_12 I6_9 I0_5 EC CEU CEM	2 4 6 1 1 1	CARRY IN SELECT SHIFT LINKAGE SELECT STATUS REG UPDATE/ COND. TEST CARRY UPDATE ENABLE USR UPDATE ENABLE MSR UPDATE ENABLE	0 0 44Q 0 0 0
CCMUX		3	COND. CODE TEST SELECT	0(UNCOND)
C2910		4	MICROSEQUENCER CONTROL	0EH(CONT)
SOF		5	SPECIAL OPERATION FIELD	0FH
IMMOP	SLWDTH	4	MIR FIELD SELECTOR WIDTH	0
	TADDR	12	TARGET ADDR. FOR DIRECT JUMP	0

3.1 Field description

The SRC field controls which register provides the source data to the ALU in double operand instructions. The DST field selects which register is read from in single and written to in single and double operand instructions. The MARS field controls the transfer of the address at the output of the ALU to the memory address register. The C2903 field selects the ALU operation to be performed and partially controls the ALU shifting operation, byte manipulations and writing of data to the destination. The C2904 field controls the carry-in to the ALU (I11_12), the linkages provided for shift and rotate instructions (I6_9), the status update and condition code testing (I0_5), and status register update inhibits (EC, CEU and CEM). The CCMUX field controls the feeding of conditional tests to the microsequencer. The C2910 field controls the program flow within the microcode. The SOF (special operations field) controls the interrupt processing, the CP main memory operations and the machine halt. The IMMOP field is used to provide the ALU with data and addresses which may be resolved at assemble-time.

Default values are provided for each field and sub-field so that a complete specification of all fields is not necessary when writing microcode (see Appendix I). The default is not necessarily a null operation. For example, the default condition for the microsequencer (C2910) is CONT which increments the micro-program counter by one to point to the next micro-instruction. Details of the default conditions are described later (see Section 5.12).

4. MICROP DEFINITIONS

The microp definition section serves to define 'macros' which may be used in the program section. The ability to nest microps means that the coding of microps may be simplified. For example, the addressing microp is invoked by the symbol @ within the ALU microps (see Appendix II). Thus the micro-assembler expects a list of addresses to follow the use of an ALU microp.

The microp definition section has been organised in the same order as the micro-instruction format. Thus the addressing register select is followed by ALU control, 2904 control, microsequencer control, special operations field control and lastly, immediate field control.

Where possible the mnemonics suggested in the ARO PROCESSOR DESIGN NOTES(ref.3) have been used. The ALU functions have been redefined to reflect the source (S_) and destination (_D) structure of the micro-instruction format. The special function mnemonics of the 2903 have been redefined to more closely reflect their operation as far as PDP-11 programmers are concerned.

5. ASSEMBLER SYNTAX

In essence, the microp definition section defines the syntax of the micro-assembler. The syntax of the program section is outlined in figure 3 with details being provided in the following sub-sections. To generate the desired microcode, one of the alternatives listed in figure 3 for each field may be selected. Figure 4 provides a list of the mnemonics recognised in the program section. The seldom used fields in the micro-instruction format are directly specified in the form FIELD=xxxxx. Labels are used extensively to make the microcode more readable and are of the form LABEL: which must be the first

entry on the line. Wherever possible the labels are derived from the PDP-11 assembler mnemonics. All labels which are left justified on the page (see Appendix III) must appear in the mapping PROM.

5.1 ALU function

The Am2903 ALU operation is divided into two classes of operation - standard functions (ALU) and special functions (SPC_FN). Figure 4 shows the correspondence between the mnemonics employed in the program section and the manufacturer's description(ref.4).

5.2 Source register definition

The register definitions must immediately follow the ALU or SPC_FN operation. Address operands may be omitted but their order must be preserved by inserting commas (eg RS,,MAR indicates the destination register is not required). Trailing operands may be omitted.

Details of the source register mnemonics are shown in figure 4. The mnemonics are on the left hand of the source register columns with the corresponding micro-instruction field values on the right. Registers R0 to R7 correspond to the PDP-11 registers. Registers R8 to R10 are reserved for later use in coding the routines to control the ARO. Registers ZERO, ONE and TWO hold the constants indicated by their names. Registers TEMP@ and TEMP are used as temporary registers within the microcode routines used to interpret each PDP-11 instruction.

The use of the symbol RS as a source register causes the addressing routine to use bits 6 to 8 of the instruction port to be used as the source register address. (This corresponds to the SS field of the PDP-11 instruction). Similarly, the use of RD as a source register causes bits 0 to 2 of the instruction port to be used as the source register address (the DD field).

The registers DR0, DR1 and DR2 are the input/output registers used to buffer data to the CP main memory. The register IMM corresponds to the micro-instruction field IMMOP and is used to enter immediate data (eg constants or target addresses) into the ALU. The use of PSW as a source register causes the processor status to be extracted from the C2904.

The symbol FLDSEL (OFFSET,WIDTH) causes portion of the instuction port to be used as the source register. The parameter OFFSET specifies the number of low order bits to be skipped. The parameter width defines the width in bits of the required field. This portion is then placed at the source input of the ALU with the unfilled high order bits set to zero.

5.3 Destination register definition

The destination register must immediately follow the source register definition delimted by a comma. The mnemonics used for the destination registers are shown in figure 4. Note that some of the micro-instruction field values for the source and destination registers are different even though the mnemonics are the same. This difference is due to the hardware configuration of the Control Processor.

The destination and source register mnemonics are the same with the exclusion of ZERO, ONE, TWO, MIR and QDST. ZERO, ONE and TWO are constants and thus are not available as a destination register. An attempt to use any of these three registers as a destination is trapped by the addressing microp @. The use of MIR causes the instruction port to be loaded with data (which may be a PDP-11 instruction) from the CP main memory. The address of this data is provided by the MAR register. The specification of QDST as a destination causes the Q register of the Am2903 to be loaded with the data at the output of the ALU.

5.4 MAR, Q register definition

The third register field controls the loading of the CP main memory address register (MAR) and the use of the Q register of the Am2903 as an alternative source. The symbol MAR causes the MAR register to be loaded with the address at the output of the ALU. The symbol QIN causes the Q register to be used as an alternative source to the ALU. Note that when QIN is invoked one source is specified by the RSN field, the destination is specified by DSN and the alternative source is the Q register. The symbol MARQ causes both MAR and QIN to be executed.

5.5 The C2904/shift control definition

This field controls the inhibiting of the register write cycle for the Am2903, and the Am2903 shifter operation and the corresponding linkages provided by the Am2904. Specifying TST inhibits the writing of data into the destination register. Note that this is the default condition if the destination register is not specified. STO and STOB cause a word and a byte, respectively, to be written into the destination register. The specification of an ALU or SPC_FN function causes the invocation of the STO operation. BSX propagates the sign of the low order byte through the high order byte and stores the result. ROR, ROL and ASL perform a similar operation to that performed by PDP-11 instructions of the same name. However, the overflow bit is not nessesarily identical. (The overflow bit is corrected by the microcode routine FIXCC - see Appendix III). Note that the Am2904 is not capable of directly simulating the ASR instruction (see microcode routine ASR3).

5.6 USR and MSR control

This group of instructions controls the manipulation of the status registers within the Am2904. The Am2904 contains two status registers - the micro status register (USR) and the machine status register (MSR). The USR is used to preserve the status of the condition codes at the microcode level. The MSR is used to preserve the status of the condition codes at the machine level. Thus the MSR contains the equivalent of the PSW of the PDP-11. The condition code bits are the sign (N) bit, the zero (Z) bit, the overflow (V) bit and the carry (C) bit.

The mnemonics MSR, MSRCI and MSRNOC cause the ALU status to be stored in the MSR 'as is', with carry inverted, and with carry inhibited (NOCarry). The mnemonic MSRZ causes the MSR to be cleared. The mnemonic USR causes the ALU status to be stored in the USR. STO_PSW causes the data at the output of the ALU to be stored in the MSR.

5.7 Transfer type

This group of instructions controls the operation of the microsequencer (C2910). Figure 4 shows the operations performed by the C2910 (see reference 4 for further details). The mnemonics JMP LABEL and JSR LABEL cause an unconditional jump (CJP UNCOND) and unconditional jump to

subroutine (CJS UNCOND), respectively. If the keyword INDEXED is used as a label for the JMP or JSR instruction, then the ALU output is used as the target address. The nmemonic RETURN is used to perform an unconditional return from a subroutine (CRTN UNCOND). NEXT implements the fetch cycle which in turn executes the JMAP instruction of the C2910. TRAP causes the microcode to enter a routine which signifies that an illegal instruction code has been detected. All other C2910 instructions take the form TRANSFR COND with the appropriate mnemonics detailed in figure 4. The conditional test mnemonics are similar to those of the PDP-11 instruction set with 'U' signifying the USR and 'M' signifying the MSR.

5.8 Memory control

This group controls the reading from (READ, READB) or writing to (WRITE, WRITEB) the CP main memory. The character B appended to the mnemonic signifies that a byte memory control operation is to be performed. In addition, READC and WRITEC read and write from the cache memory.

5.9 Loading of immediate field

The mnemonic PRESET causes the micro-assembler to expect a constant which it places in the IMMOP field. This permits constants to be used as source values. The mnemonic TADDR causes the low 12 bits of the IMMOP field to be used as a target address for the microcode.

5.10 Comments

Comments are entered into the source lines by enclosing the comment in double quote marks.

5.11 Terminator

A semicolon signifies the end of each micro-instruction.

5.12 Default field values

Default values are provided for all the microword fields to simplify the writing of the microcode (see Table 1). The destination register defaults to R0 and writing to the destination register is inhibited by the addressing microp. For example, using the addressing sequence R1,,MAR inhibits the writing of data to the destination register. The I1_4 field of the C2903 defaults to the MOV_D instruction as this simply reads the data from the destination and writes it back in again. The I5-8 sub-field of the C2903 defaults to TST which inhibits the writing of data into the destination register.

The C2904 field controls the carry-in selection, the shift linkage selection, the status updating and the condition code testing. The carry-in select defaults to inhibiting any carry-in. The I0_5 sub-field defaults to inhibiting the update of the USR or the MSR. The I0_5 sub-field also defaults to the condition code test MEQ(44Q).

The CCMUX field defaults to the unconditional mode. The C2910 (microsequencer) field defaults to the CONT instruction which causes the micro program counter to be incremented by one. The SOF field defaults to the enable interrupt condition.

6. THE PROGRAM SECTION

The program section of the microcode (see Appendix III) consists of three sections - symbol definition, subsidiary routines and the PDP-11 instruction interpretation.

6.1 The symbol definition section

The first portion of the program section serves to define registers and miscellaneous symbols used in the remainder of the program section. All of these symbols were described in figure 4 with the exception of MC and DUMMY_ADDR. MC is used to specify that the carry-in is to come from the MSR. DUMMY_ADDR specifies a non-existant location in the CP main memory which is used in transferring data from the DR0 register along the memory data bus to the instruction port (see REGDST: in the subsidiary routine section).

6.2 The subsidiary routines

This segment contains run-time routines which support the operation of the PDP-11 program section (see Appendix III). SAVE is called by the trap instructions (BPT, IOT, etc) to save the PSW and PC on the PDP-11 stack. ILGL is used to trap illegal instruction codes and traps through location 10 in CP main memory. Note that FETCH conditionally tests the interrupt status. This is the only time that the interrupt status is interrogated so interrupts are only processed at the macro level. The interrupt processing is nested so that multiple interrupts can be handled satisfactorily.

6.2.1 Address evaluation

The next group of support routines fetches the operand for PDP-11 addressing modes other than mode=0. Separate addressing routines are needed for byte addressing and for evaluating source and destination addresses. Each of the addressing groups interprets mode=1 through to mode=7. DBASE evaluates word destination addresses and DBASEB byte destination addresses. SBASE evaluates word source addresses and SBASEB byte source addresses.

In the next group of support routines, RLGDST transfers the source register value to the low order position of the instruction port. This is utilised in processing some of the PDP-11 extended insruction set codes. FIXCC is used to set the overflow bit to an identical state to the PDP-11 after arithmetic shift and rotate instructions.

6.3 PDP-11 instruction section

This portion interprets the PDP-11 instructions. It is arranged in order of increasing instruction value (eg HALT=00000Q comes first). Labels which are left justified appear in the mapping PROMS. Any labels which start with @ are used only as a jump point within each PDP-11 instruction. Occasionally @ is used within a label to distinguish it from a previously defined symbol. Note that instructions are grouped into classes. The classes are selected on the basis of likeness in starting address decoding and likeness in interpretation. For example, the miscellaneous instructions are grouped together because the low order six bits of the instruction must be evaluated to resolve the starting address.

6.3.1 Class M1 (miscellaneous)

This group includes the HALT, WAIT, RTI, BPT, IOT, RESET and RTT instructions (see Appendix III). The high order ten bits of the PDP-11 instruction are used to determine the starting address of the microcode routine which interprets each instruction. Thus if the six low order bits are necessary to interpret the instruction then the field selector must be used to resolve the starting address within each class of instructions. Note that in this case, the microcode to interpret each PDP-11 instruction must occur at a specific microcode location relative to the associated JMP INDEXED instruction. Hence if the interpretation takes more than one micro-instruction, a second jump is necessary (eg HALT: and @HALT:). Also note that attempts to use the PDP-11 RESET instuction are trapped as this instruction has no significance as far as the CP hardware is concerned. TRAP causes control to be transferred to CP main memory location 10 (octal) which is the standard PDP-11 trap location for an illegal instruction.

6.3.2 Class JUMP, RTS and CC

These three groups of instructions interpret the JMP, RTS and the Condition Code instructions(CLN, CLZ, CLV, CLC, CCC, SEN, SEZ, SEV, SEC and SCC).

The JUMP instruction utilises the field selector to generate an offset address to resolve the various modes (see Appendix III.5). The jump address is then retrieved and replaces the old value in R7. All six jump modes are interpreted.

The RTS instruction uses the field selector to obtain the register number containing the return address. The stack in PDP-11 stack is then popped to retrieve the old register value.

The CC instruction group uses the field selector to prepare a mask based on the op-code. This mask is then used to clear or set designated bits. Note that some illegal instruction codes are possible within the CC group. These are detected and control transferred to ILGL.

6.3.3 Class SWAB, BR and JSR

The function of the Control Processor is to create a monitor to control the operation of the ARO. Hence there are instructions in the PDP-11 instruction set that are used infrequently. Most byte operations fit into this category and thus there is little hardware support for byte manipulation. This makes PDP-11 instructions such as SWAB rather cumbersome as the byte swap is performed by rotating left eight times (see Appendix III.6). Note that the word to be rotated is written into a temporary register so that the same rotate routine (ROT8) can be used with both register and memory resident data.

The branch instructions (BR, BNE, BEQ, BGE, BLT, BGT and BLE) and the JSR instructions are handled in the next two groups. The JSR instructions all require the manipulation of the PDP-11 stack which is held in the CP main memory. This manipulation is performed by the microcoded routine STACK0.

6.3.4 Class S01, S02, S03, S04, S05

This group of instructions (see Appendix III.7) comprise the Single Operand instructions. Classes S01 and S02 interpret CLR, COM, INC, DEC, NEG, ADC, SBC and TST instructions. Class S01 performs register

operations (viz mode=0) and with the exception of SBC they are accomplished in one microcycle (plus the fetch cycles). Class SO2 require memory operations (viz modes other than zero) and invokes the microcode routine DBASE to obtain the destination address.

Classes SO3 and SO4 interpret ROR, ROL, ASR, ASL and SXT instructions. PDP-11 arithmetic shift and rotate operations are a little awkward to implement on the 2900 family. This arises due to the need to determine the exclusive-or (XOR) of the N and the C bits after the rotate operation. The Am2903 determines the condition codes before the rotate/shift operation is performed so the codes must be determined on a subsequent cycle. Also the Am2904 cannot directly perform a conditional jump on testing (N XOR C).

The microcode routine FIXCC is used to evaluate the condition codes after performing a shift or rotate function. The manipulation of the overflow bit is performed in the USR in preference to the MSR as the USR has individual bit control. In addition, the Am2903 is not capable of 'recycling' the sign bit as is required for ASR. These limitations slow the speed of operation of the Control Processor when handling arithmetic shift or rotate operations.

Class SO5 includes the MARK instruction which tidies the PDP-11 stack on returning from a subroutine.

6.3.5 Class DO1, DO2, DO3, DO4

This group of instructions comprises the *Double Operand instructions* (see Appendix III.8). Classes DO1, DO2, DO3 and DO4 interpret MOV, CMP, BIT, BIC, BIS and ADD. With the double operand instructions four distinct classes arise as entry points are required to separately process the occurrences of mode=0 for both the source and the destination registers. Class DO1 processes the instruction when modes of both the souce and the operand are zero. Class DO2 processes the instruction when the source mode is zero and the destination mode is not zero. Similar arrangements apply for Classes DO3 and DO4.

6.3.6 Class EIS

Class EIS interprets the Extended Instruction Set (see Appendix III.9). This set includes MUL, DIV, ASH, ASHC, XOR, FADD, FSUB, FMUL, FDIV and SOB. However FADD, FSUB, FMUL and FDIV are not interpreted but are trapped.

The MUL instruction is implemented in three stages. The pre-amble (up to the SMUL operation) serves to place the multiplier in the Q register, the multiplicand in TEMPQ, and 14 in the Am2910 loop counter. The multiplication involves fifteen cycles of SMUL (Signed MULtiplication) and one cycle of FMUL (Final MULtiplication). The FMUL operation serves to adjust the partial product depending on the sign of the multiplier. The post-amble places the 32 bit product in the designated register pair and fixes the condition codes. The MUL instruction has two entry points - one for mode=0 and the other for non zero modes.

The PDP-11 divide is integer in nature and to avoid the necessity of normalizing the dividend and divisor and post-scaling the quotient and remainder, a scheme similar to that of Rhyne(ref.5) was implemented. Rhyne's work described the case for fractional arithmetic. In the PDP-11 case (integer), the dividend (both most and least significant haves) has to be shifted left arithmetically one position before the division may proceeed. After the division algorithm has been

implemented, Rhyne suggests a correction for the quotient and remainder based on the sign bits of the quotient, the remainder and the dividend. In the integer case, it is simpler to correct the quotient and remainder on the basis of the signs of the quotient (Q), the remainder (and the divisor (X)) as shown in Table 2.

TABLE 2. QUOTIENT AND REMAINDER CORRECTION

Original signs			Required correction	
Q	R	X	Q'	X'
-	-	-	Q+1	R-X
-	-	+	Q	R
-	+	-	Q	R
-	+	+	Q+1	R-X
+	-	-	Q	R
+	-	+	Q-1	R+X
+	+	-	Q-1	R+X
+	+	+	Q	R

The DIV instruction may be conveniently broken into three sections. The pre-amble places the divisor in DR0, invokes REGDST which places the address of the dividend (least significant half) in the DD field of the instruction (which is held in the instruction port). The pre-amble then places the divisor in TEMP and the dividend (most significant half) in TEMP@. These two values are then converted to sign magnitude representation to ensure that the divisor is larger than the dividend. Note that the dividend is shifted left prior to making this test to take the 31 bit precision of the dividend into account. The divide operation is then performed. NMZD is the first divide operation and it ascertains the sign bit of the quotient. The SDIV (Signed DIVide) operation is then executed fourteen times. The FDIV (Final DIVide) operation then adjusts the quotient by forcing the least significant bit to a one. The post-amble places the quotient and remainder in the appropriate registers and implements the correction outlined in Table 2. The condition code bits are forced to be identical to that of the PDP-11.

Due to the similarity of ASH and ASHC, both of these instructions are treated by the same routine. However ASHC uses the Q register to perform the double word shift. Firstly, the source operand is placed in TEMP@ and the shift direction extracted. Then the number of shifts is placed in the C2910 loop counter. The MIR is then examined to determine if the operation pertains to ASH or ASHC and the appropriate branch made. For ASHC, the routine REGDST is invoked to determine which PDP-11 register pair holds the data to be shifted. The appropriate rotation is then performed. In the case of ASHC, a post-amble (@ASHCC) is used to determine the Z bit.

6.3.7 Class BRS and TR

Class BRS includes the BPL, BMI, BHI, BLOS, BVC, BVS, BCC, BHIS, BCS and BLO instructions. These instructions are interpreted in a similar fashion to the class BR group. Class TR includes the EMT and TRAP instructions. These are implemented similar to BPT in the class M1 group.

6.3.8 Class S06, S07, S08, S09, S010

Classes S06, S07, S08 and S09 include all the Single Operand (byte) instructions. CLRB, COMB, INCB, NEGB, ADCB and TSTB are implemented by the S06 (mode=0 and S07 (other modes) groups). Note that the lower byte only is affected for register operations. DBASEB is used to extract the byte operand for memory resident operands. In this case, either a high or low byte may be specified, but the hardware places that byte in the low position in DR0. Classes S08 (mode=0) and S09 (other modes) implement the RORB, ROLB, ASRB and ASLB instructions. The operation of these groups is similar to groups S03 and S04. Group S010 implements the MTPS and MFPS instructions.

6.3.9 Class D05, D06, D07, D08

These classes include the Double Operand (byte) instructions and interpret MOVB, CMPB, BITB, BICB and BISB. The instructions are interpreted in a similar fashion to their word counterparts in classes D01, D02, D03 and D04.

6.3.10 Class D09, D010, D011, D012

These classes interpret the SUB instruction and are similar to the ADD instruction found in D01, D02, D03 and D04.

6.4 Indirect mapping

To avoid the necessity of reburning the mapping PROMs during the debug p'ase, the entry point for each macro-instruction is indirectly mapped. Direct mapping is applied to each class of instructions. This results in the need to re-burn the mapping PROMs only if the number of instruction classes is altered. The post-processing program CPROC (see Appendix IV) examines the listing file for left justified labels and creates a mapping entry for each such label. The effect of the indirect mapping is to add one micro-cycle to the instruction timing. When the microcode is considered stable the indirect mapping may be removed.

6.5 Special macro-instructions

The 40 bit facilities of the AR0 need to be accessible at the macro level. Special macro-instructions are included in the microcode to enable the reading and writing of the cache memory and performing 40 bit data dependent processing of cache. At the time of writing, the special macro-instructions had only been partially defined. This is the area in which future expansion of the microcode is anticipated.

The instruction codes 17xxxx are used for these special macro-instructions. To facilitate using this group as they are defined, the specials are organised into eight sub-groups decoded by the field selector corresponding to codes 170xxx to 177xxx. The special macro-instructions of Appendix III.14 are tentative in nature and are included only for the sake of completeness. The special macro-instructions will be the subject of a separate document in the future.

7. PROCESSOR PERFORMANCE

7.1 Limitations and exclusions

The entire basic instruction set of the PDP-11/34 has been microcoded with the exclusion of the RESET instruction. (The RESET instruction only pertains to the operation of the UNIBUS and hence is not relevant to the hardware configuration of the Control Processor).

7.2 Processor instruction timing

The execution time for an instruction depends on the instruction itself, and on the addressing modes. In the most general case, the Instruction Execution Time is the sum of a Fetch Time, a Source Address Time, a Destination Address Time, and an Execute Time.

Instr Time = Fetch Time + SRC Time + DST Time + Exec Time

Some of the instructions require only some of these times, and are noted. All instructions require the Fetch Time which takes 3 clock cycles, and an Exec Time which takes at least 1 cycle. The number of cycles taken by each of the instructions is shown in Tables 3 to 8. The time taken for each micro-cycle is 300 ns.

7.3 Validation

The correct execution of the microcode was established by running the microcode on an emulator. The emulator has the ability to accept PDP-11 source code and execute it. To prove the correctness of the microcode, the DEC diagnostic packages CPAF and CPBG(ref.6) were run on the emulator. These two packages test the in-line code and the extended instruction set respectively.

8. ACKNOWLEDGEMENTS

The author is indebted to Mr J. Ziukelis, Dr T. Hobbs and Mr P. Drewer for information and countless discussions on the operation of the hardware.

TABLE 3. TIMING FOR MISCELLANEOUS INSTRUCTIONS

INSTRUCTION	EXEC TIME (CYCLES)
HALT	3
WAIT	4(incl 2 cyc loop)
RTI	8
BPT	11
IOT	11
RTT	9
RTS	4
CC	6
MARK	5
TRAP	5
EMT	5
BR(uncond)	3
BR(cond)	4

Instr Time = Fetch Time + Exec Time

TABLE 4. TIMING FOR JMP AND JSR INSTRUCTIONS

MODE	EXEC TIME (CYCLES)	
	JMP	JSR
1	2	6
2	3	7
3	4	8
4	3	7
5	4	8
6	4	9
7	6	11

Instr Time = Fetch Time + Exec Time

TABLE 5. TIMING FOR SINGLE OPERAND INSTRUCTIONS

INSTR	EXECUTE TIME (CYCLES)			
	WORD OPERAND		BYTE OPERAND	
	MODE=0	MODE≠0	MODE=0	MODE≠0
CLR	1	3	1	5
COM	1	3	2	5
INC	1	3	3	5
DEC	1	3	3	5
NEG	1	3	2	5
ADC	1	3	2	5
SBC	2	4	4	6
TST	1	3	1	5
ROR	6	8	8	9
ROL	6	8	7	9
ASR	7	9	9	10
ASL	6	8	7	9
SXT	2	4	2	4
	MTPS	1	3	
	MFPS	2	3	
	SWAB	10	13	

Instr Time = Fetch Time + DST Time + Exec Time
(Note that DST Time is only added if DST mode ≠ 0)

TABLE 6. TIMING FOR DESTINATION AND SOURCE OPERANDS

MODE	DST TIME (CYCLES)		SRC TIME (CYCLES)	
	WORD	BYTE	WORD	BYTE
0	0	0	0	0
1	1	1	1	1
2	2	2	2	5
3	3	3	3	3
4	1	1	1	1
5	3	3	3	3
6	3	3	3	3
7	5	5	5	5

TABLE 7. TIMING FOR EXTENDED INSTRUCTION SET

	EXECUTION TIME (CYCLES)		
	SOURCE MODE		ADD FOR EACH SHIFT POSITION
	MODE=0	MODE≠0	
MUL	32	34	-
DIV	40	42	-
ASH(left)	9	11	4
ASH(right)	7	9	2
ASHC(left)	17	19	3
ASHC(right)	17	19	2
XOR	1	3	-
SOB	4	-	-

Instr Time = Fetch Time + SRC Time + Exec Time
(Note that SRC Time is only added if DST mode ≠ 0)

TABLE 8. TIMING FOR DOUBLE OPERAND INSTRUCTIONS

INSTR	EXECUTE TIME (CYCLES)			
	SRC MODE = 0		SRC MODE ≠ 0	
	DST MODE=0	DST MODE≠0	DST MODE=0	DST MODE≠0
MOV	1	3	3	6
CMP	1	3	3	6
BIT	1	3	3	6
BIC	1	3	3	6
BIS	1	3	3	6
ADD	1	3	3	6
SUB	1	3	3	6
MOVB	4	5	5	8
CMPB	4	5	5	8
BITB	4	5	5	8
BICB	4	5	5	8
BISB	4	5	5	8

Instr Time = Fetch Time + SRC Time + DST Time + Exec Time
(Note that SRC Time is only added if SRC mode ≠ 0
and DST Time is only added if DST mode ≠ 0)

REFERENCES

No.	Author	Title
1	Ziukelis, J.	"ARO Processor - Schematic Diagrams". 18 January 1980 (unpublished work)
2	-	"Signetics Micro-assembler Reference Manuals". 8X02 AS 1000SS
3	Ziukelis, J.	"ARO Processor - Design Notes". 22 November 1979 (unpublished work)
4	-	"The Am2900 Family Data Book". Advanced Micro Devices, Inc., 1978
5	Rhyne, V.T.	"A Simple Postcorrection for Nonrestoring Division". IEEE Transactions on Computers, February 1971
6	-	"DEC/X11 General Products Module Library 1". Digital Equipment Corp. December 1976
7	-	"Build a Microcomputer". Advanced Micro Devices, Inc., 1978
8	-	"PDP11 Processor Handbook - PDP11/04/24/34A/44/70". Digital Equipment Corp. 1981
9	Ziukelis, J.	"An Overview of the ARO Processor". 14 September 1979 (unpublished work)

APPENDIX I

THE SOURCE FOR THE MICRO INSTRUCTION FORMAT

```
INSTRUCTION WIDTH 65                    "MICRO STORE WIDTH";

  FIELD SRC WIDTH 6 DEFAULT 0           "THE A REG IS THE SRC";
    FORMAT;
      FIELD AMODE WIDTH 2 DEFAULT 0     "A-MODE SELECT";
      FIELD AFIELD WIDTH 4 DEFAULT 0    "A-SOURCE SELECT";
    END FORMAT;
  FIELD DST WIDTH 5 DEFAULT 27Q         "THE B REG IS THE DST";
    FORMAT;
      FIELD BMODE WIDTH 1 DEFAULT 1     "B-MODE SELECT";
      FIELD BFIELD WIDTH 4 DEFAULT 7    "B-SOURCE SELECT";
    END FORMAT;
  FIELD MARS WIDTH 1 DEFAULT 0          "MEM ADDR REG SELECT";
  FIELD C2903 WIDTH 9 DEFAULT 8CH       "2903 CONTROL";
    FORMAT                              "2903 FIELDS, DEFAULT TO MOVD OP";
      FIELD I1_4 WIDTH 4 DEFAULT 4      "ALU OPN AND SPECIAL FNS";
      FIELD I3 WIDTH 1 DEFAULT 0        "ALU OPN";
      FIELD I5_8 WIDTH 4 DEFAULT 0CH    "ALU DEST CNTRL INC SHIFT,SPEC";
    END FORMAT;
  FIELD C2904 WIDTH 15 DEFAULT 6440Q    "2904 CONTROL";
    FORMAT                              "2904 FIELDS";
      FIELD I11_12 WIDTH 2 DEFAULT 0    "ALU CARRY IN SEL, DEF=INHIBIT";
      FIELD I6_9 WIDTH 4 DEFAULT 6      "SHIFT LINKAGE SEL";
      FIELD I0_5 WIDTH 6 DEFAULT 44Q    "STATUS REG UPDATE AND TEST";
      FIELD EC WIDTH 1 DEFAULT 0        "CARRY UPDATE, DEF=INHIBIT";
      FIELD CEU WIDTH 1 DEFAULT 0       "USR UPDATE, DEF=INHIBIT";
      FIELD CEM WIDTH 1 DEFAULT 0       "MSR UPDATE, DEF=INHIBIT";
    END FORMAT;
  FIELD CCMUX WIDTH 3 DEFAULT 0         "TEST CC SELECT, DEF=UNCOND";
  FIELD C2910 WIDTH 4 DEFAULT 0EH       "2910 CONTROL, DEF=CONTINUE";
  FIELD SOF WIDTH 5 DEFAULT 0FH         "SPECIAL OPERATION FIELD
                                         DEF=ENABLE INTERRUPT";
  FIELD IMMOP WIDTH 16 DEFAULT 0        "IMMEDIATE FIELD";
    FORMAT                              "DIRECT CJUMP";
      FIELD SLWDTH WIDTH 4 DEFAULT 0    "MIR FIELD SELECTOR WIDTH";
      FIELD TADDR WIDTH 12 DEFAULT 0    "TARGET ADDR FOR DIR JUMP";
    END FORMAT;
  FIELD ERROR WIDTH 1 DEFAULT 0         "SET IF ERROR";
END INSTRUCTION;
```

APPENDIX II

THE SOURCE FOR THE MICROP DEFINITION SECTION

```
"THE MICROP DEFINITION SECTION"

   II.1  The address interpreter

     "ADDRESS INTERPRETER - INVOKED BY 2903 ALU AND SPECIAL OPERATIONS"

     MICROP @ RSN(OFFSET,WDTH),RDN,TMAR
       ASSIGN                          "SET UP SRC,DST FOR 2903"
       "DEFINITION OF LAST FIELD(TMAR)"
         IF (TMAR EQ 'MAR')
           THEN                        "WRITE TO MAR"
             MARS=1
           ELSE
             IF (TMAR EQ 'QIN')
               THEN                    "READ FROM Q"
                 I0=1
               ELSE
                 IF (TMAR EQ 'MARQ')
                   THEN                "WRITE TO MAR, READ FROM Q"
                     MARS=1 I0=1
                   ELSE
                 FI
             FI
         FI

         "DEFINITION OF SECOND FIELD(DESTINATION)"
         IF (RDN EQ 'ZERO') OR (RDN EQ 'ONE') OR (RDN EQ 'TWO')
           THEN                        "FLAG CATASTROPHIC ERROR"
             ERROR=3                   "ATTEMPT TO OVERWRITE CONSTANTS"
           ELSE
         FI
         IF (RDN GE 0) AND (RDN LE 15) OR (RDN EQ 15H) OR (RDN EQ 27Q)
           THEN                        "DST IS GP CPU REG"
             DST=RDN
           ELSE
               IF (RDN GE 20H) AND (RDN LE 23H)
                 THEN                  "REDEFINE DEDICATED CPU REG"
                   DST=RDN-10H
                 ELSE
                   IF RDN EQ 'RD'
                     THEN              "DST IS DEFINED BY DD FIELD"
                       DST=30Q
                     ELSE
                       IF RDN EQ 'RS'
                         THEN          "DST IS DEFINED BY SS FIELD"
                           DST=34Q
                         ELSE
                           IF (RDN EQ 'QDST')
                             THEN      "DST IS Q REG"
                               I5_8=6  "STORE F IN Q"
                             ELSE      "INHIBIT RAM WRITE CYCLE"
                               I5_8=0CH
                           FI
                       FI
                   FI
               FI
```

```
    FI

  "DEFINITION OF FIRST FIELD(SOURCE)"
  IF WDTH GT 0                          "TEST FOR FIELD SEL OPTION"
    THEN                                "SET UP FIELD SEL"
      AMODE=3 AFIELD=OFFSET SLWDTH=16-WDTH
    ELSE
      IF RSN EQ 'RD'
        THEN                            "SRC SEL FROM DD FIELD"
          SRC=18H
        ELSE
          IF RSN EQ 'PSW'
            THEN                        "SRC IS PSW (ALU INHIBITED)"
              SOF=6 I0_5=40Q
            ELSE
              IF RSN EQ 'RS'
                THEN                    "SRC IS DEFINED BY SS FIELD"
                  SRC=10H
                ELSE
                  SRC=RSN               "SRC IS GP OR DEDICATED CPU REG"
              FI          "SEE ASSEMBLER DIRECTIVES FOR DEFINITION"
          FI
      FI
    FI;
```

II.2 The Am2903 microps

```
"MNEMONICS SUGGESTED BY J. ZIUKELIS"
"ALU MICROPS - CONTROLS 2903 - DEFAULTS TO STORE WORD MODE"
  MICROP SET_F RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0              DEFAULT I5_8=4 "SET ALU OUTPUT";
  MICROP D_SUB_S RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=1 I11_12=1     DEFAULT I5_8=4 "DST-SRC";
  MICROP S_SUB_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=2 I11_12=1     DEFAULT I5_8=4 "SRC-DST";
  MICROP S_ADD_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=3              DEFAULT I5_8=4 "SRC+DST";
  MICROP MOVD RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=4              DEFAULT I5_8=4 "DST TO F";
  MICROP NOT_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=5              DEFAULT I5_8=4 "NOT DST";
  MICROP MOVS RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=6              DEFAULT I5_8=4 "SRC TO F";
  MICROP NOT_S RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=7              DEFAULT I5_8=4 "NOT SRC";
  MICROP CLR RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=8              DEFAULT I5_8=4 "CLR ALU OUTPUT";
  MICROP NS_AND_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=9              DEFAULT I5_8=4 "NOT SRC AND DST";
```

```
  MICROP S_NXOR_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0AH          DEFAULT I5_8=4 "SRC NXOR DST";
  MICROP S_XOR_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0BH          DEFAULT I5_8=4 "SRC XOR DST";
  MICROP S_AND_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0CH          DEFAULT I5_8=4 "SRC AND DST";
  MICROP S_NOR_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0DH          DEFAULT I5_8=4 "SRC NOR DST";
  MICROP S_NAND_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0EH          DEFAULT I5_8=4 "SRC NAND DST";
  MICROP S_OR_D RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I1_4=0FH          DEFAULT I5_8=4 "SRC OR DST";


"SPC_FN MICROPS - CONTROLS 2903"
  MICROP MUL RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=0                "MULT OP";
  MICROP SMUL RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=2                "SIGN MULT";
  MICROP INC1 RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=4                "INC BY 1";
  MICROP INC2 RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=4 I11_12=1       "INC BY 2";
  MICROP CONV RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=5                "SIGN MAG TO TWO COMP";
  MICROP MULF RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=6                "MULT, LAST CYCLE";
  MICROP NMZS RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=8                "NORM SINGLE PREC";
  MICROP NMZD RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=0AH              "NORM DOBLE PREC";
  MICROP SDIV RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=0CH              "SIGN DIVIDE";
  MICROP FDIV RSN=0(OFFSET=0,WDTH=0),RDN=99,MAD=0
    ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD
    I0=0 I1_4=0 I5_8=0EH              "SIGN DIV, LAST CYC";
```

II.3 The Am2904 microps

```
"SHFT MICROPS - CONTROLS 2903, 2904"

  MICROP TST ASSIGN I5_8=0CH I6_9=6  "TST OP, INHIBITS WRITE";
  MICROP STO ASSIGN I5_8=4 I6_9=6    "STO OP, STORES WORD";
  MICROP STOB ASSIGN I5_8=0FH I6_9=6 "STOB OP, STORES BYTE";
```

```
  MICROP BSX ASSIGN  I6_9=6
                     I5_8=0EH        "BSX OP, BYTE SIGN EXTEND";
  MICROP ROR ASSIGN I5_8=1 I6_9=9    "ROR OP, AS PER PDP11";
  MICROP ROL ASSIGN I5_8=9 I6_9=9    "ROL OP, AS PER PDP11";
  MICROP ASL ASSIGN I5_8=9 I6_9=0    "ASL, NOTE SIGN BIT NOT RELOADED";


"STATUS TEST OPTIONS"

  MICROP ULE ASSIGN CCMUX=1 IO_5=20Q "USR LE";
  MICROP UGT ASSIGN CCMUX=1 IO_5=21Q "USR GT";
  MICROP ULT ASSIGN CCMUX=1 IO_5=22Q "USR LT";
  MICROP UGE ASSIGN CCMUX=1 IO_5=23Q "USR GE";
  MICROP UEQ ASSIGN CCMUX=1 IO_5=24Q "USR EQ";
  MICROP UNE ASSIGN CCMUX=1 IO_5=25Q "USR NE";
  MICROP UVS ASSIGN CCMUX=1 IO_5=26Q "USR V SET";
  MICROP UVC ASSIGN CCMUX=1 IO_5=27Q "USR V CLEAR";
  MICROP UCS ASSIGN CCMUX=1 IO_5=32Q "USR C SET";
  MICROP UCC ASSIGN CCMUX=1 IO_5=33Q "USR C CLEAR";
  MICROP ULOS ASSIGN CCMUX=1 IO_5=34Q "USR LOWER OR SAME";
  MICROP UHI ASSIGN CCMUX=1 IO_5=35Q "USR HIGHER";
  MICROP UMI ASSIGN CCMUX=1 IO_5=36Q "USR MINUS";
  MICROP UPL ASSIGN CCMUX=1 IO_5=37Q "USR PLUS";
  MICROP MLE ASSIGN CCMUX=1 IO_5=40Q "MSR LE";
  MICROP MGT ASSIGN CCMUX=1 IO_5=41Q "MSR GT";
  MICROP MLT ASSIGN CCMUX=1 IO_5=42Q "MSR LT";
  MICROP MGE ASSIGN CCMUX=1 IO_5=43Q "MSR GE";
  MICROP MEQ ASSIGN CCMUX=1 IO_5=44Q "MSR EQ";
  MICROP MNE ASSIGN CCMUX=1 IO_5=45Q "MSR NE";
  MICROP MVS ASSIGN CCMUX=1 IO_5=46Q "MSR V SET";
  MICROP MVC ASSIGN CCMUX=1 IO_5=47Q "MSR V CLEAR";
  MICROP MCS ASSIGN CCMUX=1 IO_5=52Q "MSR C SET";
  MICROP MCC ASSIGN CCMUX=1 IO_5=53Q "MSR C CLEAR";
  MICROP MLOS ASSIGN CCMUX=1 IO_5=50Q "MSR LOWER OR SAME";
  MICROP MHI ASSIGN CCMUX=1 IO_5=51Q "MSR HIGHER";
  MICROP MMI ASSIGN CCMUX=1 IO_5=56Q "MSR MINUS";
  MICROP MPL ASSIGN CCMUX=1 IO_5=57Q "MSR PLUS";


"STATUS UPDATE MICROPS - CONTROL 2904"

  MICROP MSR ASSIGN IO_5=44Q CEM=1 EC=1 "LOAD MSR";
  MICROP MSRCI ASSIGN IO_5=30Q CEM=1 EC=1 "LOAD MSR WITH CARRY INV";
  MICROP MSRNOC ASSIGN IO_5=44Q CEM=1 "LOAD MSR WITH NO CARRY";
  MICROP MSRZ ASSIGN IO_5=3 CEM=1 EC=1 "CLEAR MSR";
  MICROP USR ASSIGN IO_5=44Q CEC=1   "LOAD USR";
  MICROP STO_PSW ASSIGN IO_5=0 EC=1 CEM=1
              SOF=11Q  I5_8=4        "TRANSFER Y TO MSR";


"CCMUX CONTROL MICROPS"

  MICROP UNCOND ASSIGN CCMUX=0       "UNCOND JUMP-DEFAULT";
  MICROP INTREQ ASSIGN CCMUX=2       "INTERRUPT REQUEST RECEIVED";



II.4 Microsequencer microps

"TRSNFR MICROPS - CONTROLS 2910"
```

```
  MICROP JZ ASSIGN C2910=0          "JUMP ZERO";
  MICROP CJS ASSIGN C2910=1         "COND JUMP SUBR";
  MICROP JMAP ASSIGN C2910=2        "JUMP TO MAP";
  MICROP CJP ASSIGN C2910=3         "COND JUMP UIR";
  MICROP PUSH ASSIGN C2910=4        "PUSH, COND LOAD CNTR";
  MICROP JSRP ASSIGN C2910=5        "COND JUMP SUBR, CNTR OR UIR";
  MICROP CJV ASSIGN C2910=6         "COND JUMP VECTOR";
  MICROP JRP ASSIGN C2910=7         "COND JUMP CNTR OR UIR";
  MICROP RFCT ASSIGN C2910=8        "REPEAT LOOP";
  MICROP RPCT ASSIGN C2910=9        "REPEAT UIR";
  MICROP CRTN ASSIGN C2910=0AH      "COND RETURN";
  MICROP CJPP ASSIGN C2910=0BH      "COND JUMP UIR";
  MICROP LDCT ASSIGN C2910=0CH      "LOAD COUNTER";
  MICROP LOOP ASSIGN C2910=0DH      "TEST END OF LOOP";
  MICROP CONT ASSIGN C2910=0EH      "USE NEXT UIR";
  MICROP TWB ASSIGN C2910=0FH       "THREE WAY BRANCH";
```

II.5 Subsidiary microps

```
"SUBSIDIARY MICROPS"

  "JMP DEFINITION - JUMPS UNCONDITIONALLY TO ANY 'LABEL'
    IF 'INDEXED' IS LABEL THEN IMMEDIATE FIELD USED AS JUMP ADDR"
  MICROP JMP LABEL ASSIGN CJP UNCOND
    IF (LABEL NE 'INDEXED')
      THEN                          "ASSIGN LABEL"
        TADDR=LABEL
      ELSE                          "ADDRESS ALREADY CALC"
    FI;

  "JSR DEFINITION - PUSHES STACK AND UNCONDITIONALLY JUMPS TO 'LABEL'
    IF 'INDEXED' IS LABEL THEN IMMEDIATE FIELD USED AS JUMP ADDR"
  MICROP JSR LABEL ASSIGN CJS UNCOND
    IF (LABEL NE 'INDEXED')
      THEN                          "ASSIGN LABEL"
        TADDR=LABEL
      ELSE                          "ADDRESS ALREADY CALC"
    FI;

  MICROP RETURN ASSIGN CRTN UNCOND  "UNCOND RETURN";
  MICROP NEXT ASSIGN JMP FETCH      "JUMP TO MACRO FETCH ROUTINE";
  MICROP TRAP ASSIGN ERROR=1 JMP ILGL "ILLEGAL INSTRUCTION";
```

II.6 Memory and immediate field microps

```
"MEMORY CONTROL MICROPS"

  MICROP READ ASSIGN SOF=10H        "SET MEM CNTRL BITS FOR READ";
  MICROP WRITE ASSIGN SOF=14H       "SET MEM CNTRL BITS FOR WRITE";
  MICROP READB ASSIGN SOF=11H       "SET MEM CNTL FOR BYTE READ";
  MICROP WRITEB ASSIGN SOF=15H      "SET MEM CNTL FOR BYTE WRITE";
  MICROP READC ASSIGN SOF=13H       "SET MEM CNTL FOR CACHE READ";
  MICROP WRITEC ASSIGN SOF=17H      "SET MEM CNTL FOR CACHE WRITE";

"IMMEDIATE FIELD CONTROL MICROPS"
```

```
MICROP PRESET=VALUE ASSIGN IMMOP=VALUE "PRESET IMMEDIATE FIELD";
```

APPENDIX III

THE SOURCE FOR THE PROGRAM DEFINITION SECTION

III.1 The symbol definition section

```
PROGRAM TEST WIDTH 65 LENGTH 400H;


"THE PROGRAM DEFINITION SECTION"
"SYMBOL DEFINITIONS FOR THE ASSEMBLER"

"REG DEFN FOR RSN (VIZ- SRC REG)"
  R0:       EQU     0;
  R1:       EQU     1;
  R2:       EQU     2;
  R3:       EQU     3;
  R4:       EQU     4;
  R5:       EQU     5;
  R6:       EQU     6;
  SP:       EQU     6;
  R7:       EQU     7;
  PC:       EQU     7;
  R8:       EQU     8;
  R9:       EQU     9;
  R10:      EQU     10;
  ZERO:     EQU     11 "LOAD WITH 0 AT BOOT-UP";
  ONE:      EQU     12 "LOAD WITH 1 AT BOOT-UP";
  TWO:      EQU     13 "LOAD WITH 2 AT BOOT-UP";
  TEMP@:    EQU     14;
  TEMP:     EQU     15;
  RS:       EQU     'RS';
  PSW:      EQU     'PSW';
  RD:       EQU     'RD';
  DR0:      EQU     20H;
  DR1:      EQU     21H;
  DR2:      EQU     22H;
  IMM:      EQU     23H;
  DUMP:     EQU     27Q;
  FLDSEL:   EQU     'FLDSEL';
  MAR:      EQU     'MAR';
  QDST:     EQU     'QDST';
  QIN:      EQU     'QIN';
  MARQ:     EQU     'MARQ';
  INDEXED:  EQU     'INDEXED';
  MC:       EQU     44Q;
  MIR:      EQU     14H;
  OPREG:    EQU     15H;
  DUMMY_ADDR: EQU 177176Q;
```

III.2 The subsidiary routines

```
"SUBSIDIARY ROUTINES - NOT DIRECTLY CALLED BY MAPPING PROMS"

"INTERRUPT VECTOR LOCATIONS"

  (10):    MOVS TEMP,DUMP STO_PSW JMP ODDWORD "ODD WORD ADDRESS";
  (11):    MOVS TEMP,DUMP STO_PSW JMP HLTRQ "HALT REQUEST VECTOR";
  (12):    MOVS TEMP,DUMP STO_PSW JMP DCERR "DATA CHANNEL ERROR";
```

```
(13):   MOVS TEMP,DUMP STO_PSW JMP DCRDY "DATA CHANNEL READY";
(14):   MOVS TEMP,DUMP STO_PSW JMP APINTCOND "ARITH PROC TRAP";
(15):   MOVS TEMP,DUMP STO_PSW JMP APINTRDY "ARITH PROC READY";

        ORG   10H          "START SUBSIDIARY ROUTINES AT LOC 16";


"SAVE PSW, PC ON PDP11 STACK"

SAVE:   MOVS PSW,DR0                    "DR0=PSW, MORE";
        D_SUB_S TWO,R6,MAR  WRITE       "R6=R6-2, PUSH TO STACK, MORE";
        D_SUB_S TWO,R6,MAR              "R6=R6-2, MORE";
        MOVS R7,DR0  WRITE  RETURN      "DR0=PC, PUSH TO STACK, END";


"ILLEGAL INSTUCTION TRAP"

ILGL:   MOVS IMM,TEMP  PRESET=10Q       "VECTOR FOR ILLEGAL INSTR";
        JMP GTVCTR                      "GET NEW PC,PS";
        CONT                            "DUMMY TO PRESERVE UAR";
        CONT                            "DUMMY TO PRESERVE UAR";


"MACRO FETCH ROUTINE, (PRESERVE UNA=31 FOR FETCH)"

FCHSPC: MOVS R7,,MAR  READ  JMP @FETCH "FETCH NO INTRPT";
FETCH:  MOVS R7,,MAR  READ             "MAR=PC"
          CJP INTREQ  TADDR=INTROUT    "JUMP TO INTERRUPT HANDLER IF
                                        INTERRUPT PRESENT";
@FETCH: DST=MIR                        "LOAD MIR, MORE";
        INC2 ,R7  JMAP                 "INC PC, JUMP TO START ADDR";


"SUBSIDIARY ROUTINES(CONTINUED)"

"GET NEW PC,PS POINTED TO BY TEMP"
GTVCTR:  "OLD PC,PS SAVED ON STACK"
         "PC=(TEMP),PS=(TEMP+2)"
         "PRESERVE UNA=34 FOR GTVCTR"
        JSR SAVE                       "SAVE OLD PC,PS ON STACK";
        MOVS TEMP,,MAR  READ           "RETRIEVE NEW PC";
        INC2 ,TEMP                     "POINT TO NEW PS";
        MOVS DR0,R7                    "PC=(TEMP)";
        MOVS TEMP,,MAR  READ           "RETRIEVE NEW PS";
        CONT                           "WAIT";
        MOVS DR0,DUMP STO_PSW  NEXT    "PS=(TEMP+2), END";


"INTERRUPT HANDLER ROUTINE"

INTROUT:  DST=MIR                      "DUMMY PRESERVES DR";
        MOVS PSW,TEMP                  "SAVE PSW, MORE";
        SOF=5  CJV UNCOND              "ENABLE VECTOR, JUMP TO DEFINE
                                        INTR TYPE, JUMP BACK TO INTR1";


"MACRO INTERRUPT HANDLER"
"HLTRQ IN INVOKED FROM HOST BY SETTING M-BIT IN INTERFACE"

HLTRQ:  MOVS IMM,TEMP  PRESET=240Q     "VECTOR VIA 240Q";
        JMP GTVCTR                     "JUMP TO VECTOR";
```

```
ODDWORD: MOVS IMM,TEMP  PRESET=4Q  "VECTOR VIA 4Q";
         JMP GTVCTR                "JUMP TO VECTOR";
DCERR:   MOVS IMM,TEMP  PRESET=110Q  "VECTOR VIA 110Q";
         JMP GTVCTR                "JUMP TO VECTOR";
DCRDY:   MOVS IMM,TEMP  PRESET=114Q  "VECTOR VIA 114Q";
         JMP GTVCTR                "JUMP TO VECTOR";
APINTCOND: MOVS IMM,TEMP  PRESET=120Q  "VECTOR VIA 120Q";
         JMP GTVCTR                "JUMP TO VECTOR";
APINTRDY: MOVS IMM,TEMP  PRESET=124Q  "VECTOR VIA 124Q";
         JMP GTVCTR                "JUMP TO VECTOR";


"REGDST - ROUTINE TO PLACE (MIR6_8 OR 1) IN MIR0_2
   VIZ SS OR 1  IN DD FIELD
 THIS IS USED AS REG DST IN MUL, DIV, ASHC INSTR
 TECNIQUE- FIELD SELECTOR GENERATES SSS000SS1, LOAD INTO MIR
   VIA DUMMY WRITE CYCLE"

REGDST: MOVS IMM,DR0  PRESET=700Q   "DR0=RS MASK, MORE";
        S_AND_D FLDSEL(0,9),DR0     "DR0=SSS000000, MORE";
        S_OR_D FLDSEL(6,3),DR0      "DR0=SSS000SSS, MORE";
        S_OR_D ONE,DR0              "DR0=SSS000SS1, MORE";
        MOVS IMM,,MAR  PRESET=DUMMY_ADDR
          WRITE                     "MAR=DUMMY ADDR,END";
        DST=MIR  RETURN             "MIR=DR0,END";


"FIXES OVERFLOW BIT AFTER ROTATES/SHIFTS "

FIXCC:  IO_5=16Q CEU=1              "RESET UV";
        UMI CJP TADDR=NSET          "JMP IF UN SET";
  NCLR: MCS CJP TADDR=SETV          "JMP IF MC SET(AND UN CLR)";
  XIT:  IO_5=2 CEM=1 CEU=1  NEXT    "SWAP REG(MC RETAIN), END";
  NSET: MCS CJP TADDR=XIT           "EXIT IF MC SET(AND UN SET)";
  SETV: IO_5=17Q CEU=1 JMP XIT      "SET UV AND EXIT";
```

III.3 Address evaluation

```
"ADDRESS EVALUATION ROUTINE - WORD INSTRUCTIONS"
"DD FIELD"

DBASE:  TRAP                        "MODE=0 IS ILLEGAL";
        MOVS RD,,MAR  READ  RETURN  "MODE=1, FETCH OPND, END";
        MOVS RD,,MAR  READ  JMP DD2 "MODE=2, FETCH OPND, MORE";
        MOVS RD,,MAR  READ  JMP DD3 "MODE=3, FETCH ADDR, MORE";
        D_SUB_S TWO,RD,MAR  READ  RETURN "MODE=4, DEC RD, FETCH OPND";
        D_SUB_S TWO,RD,MAR  READ  JMP DD5 "MODE=5,DEC RD, FETCH ADDR";
        MOVS R7,,MAR  READ  JMP DD6 "MODE=6, FETCH INDEX, MORE";
        MOVS R7,,MAR  READ  JMP DD7 "MODE=7, FETCH INDEX, MORE";
  DD2:  INC2 ,RD  RETURN            "AUTO INC, END";
  DD3:  INC2 ,RD                    "AUTO INC, MORE";
        MOVS DR0,,MAR  READ  RETURN "FETCH OPND, END";
  DD5:  CONT                        "WAIT FOR MEM, MORE";
        MOVS DR0,,MAR  READ  RETURN "FETCH OPND, END";
  DD6:  INC2 ,R7                    "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DR0,RD,MAR  TST  READ  RETURN "MAR=PC+INDEX+4, END";
  DD7:  INC2 ,R7                    "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DR0,RD,MAR  TST  READ "MAR=PC+INDEX+4, MORE";
        CONT                        "WAIT FOR MEM  ";
```

```
        MOVS DRO,,MAR  READ  RETURN "RD=(PC+INDEX+4), FETCH OPND,END";


"ADDRESS EVALUATION ROUTINE - BYTE INSTRUCTIONS"
"ONLY CHANGE IS TO MODE=2 AND 4"

DBASEB: TRAP                          "MODE=0 IS ILLEGAL";
        MOVS RD,,MAR  READB  RETURN "MODE=1, FETCH OPND, END";
        MOVS RD,,MAR  READB  JMP DB2 "MODE=2, FETCH OPND, MORE";
        MOVS RD,,MAR  READ   JMP DB3 "MODE=3, FETCH ADDR, MORE";
        MOVS FLDSEL(0,3),TEMP@ JMP DB4 "MODE=4, TEMP@=REG NO, MORE"; ;
        D_SUB_S TWO,RD,MAR  READ JMP DB5 "MODE=5,DEC RD, FETCH ADDR";
        MOVS R7,,MAR  READ   JMP DB6 "MODE=6, FETCH INDEX, MORE";
        MOVS R7,,MAR  READ   JMP DB7 "MODE=7, FETCH INDEX, MORE";
  DB2:                                "INC2 IF R6, R7 RELATIVE INSTR"
        MOVS FLDSEL(0,3),TEMP@        "TEMP@=DST REG NO,MORE";
        D_SUB_S IMM,TEMP@ USR  PRESET=6 "TEMP@ GE 0 FOR R6,R7";
        INC1 ,RD  CRTN ULT            "RET IF LT 0, MORE";
        INC1 ,RD  RETURN              "INC1 IF R6,R7 REL, END";
  DB3:  INC2 ,RD                      "AUTO INC, MORE";
        MOVS DRO,,MAR  READB  RETURN "FETCH OPND, END";
  DB4:                                "DEC BY 2 IF R6, R7 REL"
        D_SUB_S IMM,TEMP@ USR PRESET=6 "TEMP@ GE 0 FOR R6,R7";
        D_SUB_S ONE,RD,MAR READB CRTN ULT "RETURN IF NOT R6,R7";
        D_SUB_S ONE,RD  RETURN        "RETURN FOR R6,R7";
  DB5:  CONT                          "WAIT FOR MEM, MORE";
        MOVS DRO,,MAR  READB  RETURN "FETCH OPND, END";
  DB6:  INC2 ,R7                      "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DRO,RD,MAR  TST  READB  RETURN "MAR=PC+INDEX+4, END";
  DB7:  INC2 ,R7                      "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DRO,RD,MAR  TST  READ  "MAR=PC+INDEX+4, MORE";
        CONT                          "WAIT FOR MEM  ";
        MOVS DRO,,MAR  READB RETURN "RD=(PC+INDEX+4), FETCH OPND,END";


"SS FIELD ADDRESS EVALUATION"

SBASE:  TRAP                          "MODE=0 IS ILLEGAL";
        MOVS RS,,MAR  READ  RETURN  "MODE=1, FETCH OPND, END";
        MOVS RS,,MAR  READ  JMP SS2 "MODE=2, FETCH OPND, MORE";
        MOVS RS,,MAR  READ  JMP SS3 "MODE=3, FETCH ADDR, MORE";
        D_SUB_S TWO,RS,MAR  READ  RETURN "MODE=4, DEC RS, FETCH OPND";
        D_SUB_S TWO,RS,MAR  READ  JMP SS5 "MODE=5,DEC RS, FETCH ADDR";
        MOVS R7,,MAR  READ  JMP SS6 "MODE=6, FETCH INDEX, MORE";
        MOVS R7,,MAR  READ  JMP SS7 "MODE=7, FETCH INDEX, MORE";
  SS2:  INC2 ,RS  RETURN              "AUTO INC, END";
  SS3:  INC2 ,RS                      "AUTO INC, MORE";
        MOVS DRO,,MAR  READ  RETURN "FETCH OPND, END";
  SS5:  CONT                          "WAIT FOR MEM, MORE";
        MOVS DRO,,MAR  READ  RETURN "FETCH OPND, END";
  SS6:  INC2 ,R7                      "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DRO,RS,MAR  TST  READ  RETURN "MAR=PC+INDEX+4, END";
  SS7:  INC2 ,R7                      "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DRO,RS,MAR  TST  READ "MAR=PC+INDEX+4, MORE";
        CONT                          "WAIT FOR MEM  ";
        MOVS DRO,,MAR  READ  RETURN "RD=(PC+INDEX+4), FETCH OPND,END";


"ADDRESS EVALUATION ROUTINE - BYTE INSTRUCTIONS"
"ONLY CHANGE IS TO MODE=2 AND 4"
```

```
SBASEB: TRAP                              "MODE=0 IS ILLEGAL";
        MOVS RS,,MAR  READB  RETURN "MODE=1, FETCH OPND, END";
        MOVS RS,,MAR  READB  JMP SB2 "MODE=2, FETCH OPND, MORE";
        MOVS RS,,MAR  READ   JMP SB3 "MODE=3, FETCH ADDR, MORE";
        MOVS FLDSEL(6,3),TEMP@ JMP SB4 "MODE=4, TEMP@=REG NO, MORE"; ;
        D_SUB_S TWO,RS,MAR  READ  JMP SB5 "MODE=5,DEC RS, FETCH ADDR";
        MOVS R7,,MAR  READ   JMP SB6 "MODE=6, FETCH INDEX, MORE";
        MOVS R7,,MAR  READ   JMP SB7 "MODE=7, FETCH INDEX, MORE";
  SB2:                                    "INC2 IF R6, R7 RELATIVE INSTR"
        MOVS FLDSEL(6,3),TEMP@            "TEMP@=SRC REG NO,MORE";
        D_SUB_S IMM,TEMP@ USR  PRESET=6 "TEMP@=TEST VAL,MORE";
        INC1 ,RS  CRTN ULT                "RETURN IF NOT R6,R7,MORE";
        INC1 ,RS  RETURN                  "RETURN IF R6,R7 REL, END";
  SB3:  INC2 ,RS                          "AUTO INC, MORE";
        MOVS DR0,,MAR  READB  RETURN "FETCH OPND, END";
  SB4:  D_SUB_S IMM,TEMP@ USR PRESET=6 "TEMP@ GE 0 IF R6,R7";
        D_SUB_S ONE,RS,MAR READB CRTN ULT "RETURN IF NOT R6,R7";
        D_SUB_S ONE,RS  RETURN            "RETURN IF R6,R7, END";
  SB5:  CONT                              "WAIT FOR MEM, MORE";
        MOVS DR0,,MAR  READB  RETURN "FETCH OPND, END";
  SB6:  INC2 ,R7                          "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DR0,RS,MAR  TST  READB  RETURN "MAR=PC+INDEX+4, END";
  SB7:  INC2 ,R7                          "INC PC(VIZ ORIG PC+4) , MORE";
        S_ADD_D DR0,RS,MAR  TST  READ  "MAR=PC+INDEX+4, MORE";
        CONT                              "WAIT FOR MEM  ";
        MOVS DR0,,MAR  READB RETURN "RS=(PC+INDEX+4), FETCH OPND,END";
```

III.4 Class M1 (miscellaneous)

```
"PDP 11 INSTRUCTION DESCRIPTION"
"ALL LABELS WHICH ARE LEFT JUSTIFIED APPEAR IN MAPPING PROM"
"MISCELLANEOUS INSTRUCTIONS - CLASS M1"
"HALT, WAIT, RTI, BPT, IOT, RESET, RTT"

M1:                          "CLASS START ADDR FROM MAPPING PROM"
        S_ADD_D FLDSEL(0,3),IMM
          TADDR=$+1  JMP INDEXED          "JUMP TO M1 INSTR";
  HALT: SOF=30Q  NEXT                     "HALT FOR HOST TO CLEAR";
  WAIT: JMP @WAIT                         "LOOP WAITING FOR INTERRUPT";
  RTI:                          "RETURN FROM INTERRUPT OR TRAP INSTR"
        MOVS R6,,MAR  READ JMP @RTI "POP PC FROM STACK, MORE";
  BPT:  JMP @BPT                          "BREAK POINT INSTR";
  IOT:  JMP @IOT                          "IOT INSTR";
  RESET: TRAP                             "RESET MEANINGLESS";
  RTT:  JMP RTI                           "RTT SAME AS RTI";
  @WAIT: CJP INTREQ  TADDR=INTROUT        "TEST FOR INTERRUPT, MORE";
        JMP @WAIT                         "LOOP TO TEST AGAIN";
  @RTI: CONT                              "WAIT FOR NEW PC, MORE";
        MOVS DR0,R7                       "RESTORE PC, MORE";
        INC2 ,R6,MAR  READ                "POP PSW FROM STACK, MORE";
        CONT                              "WAIT FOR NEW PSW, MORE";
        MOVS DR0,DUMP  STO_PSW            "RESTORE PSW, MORE";
        INC2 ,R6  JMP FCHSPC              "FIX STACK POINTER, END";
  @BPT: MOVS IMM,TEMP  PRESET=14Q         "VECTOR VIA 14Q";
        JMP GTVCTR                        "GET NEW PC,PS";
  @IOT: MOVS IMM,TEMP  PRESET=16Q         "VECTOR VIA 16Q";
        JMP GTVCTR                        "GET NEW PC,PS";
```

III.5 Class JUMP,RTS,CC

```
"JMP INSTRUCTION -CLASS JUMP"

JUMP:                                   "CLASS START ADDR FROM MAP"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=$+1 JMP INDEXED       "DD MODE GIVES OFFSET, MORE";
          TRAP                          "MODE=0 IS ILLEGAL, END";
          MOVS RD,R7  NEXT              "MODE=1, PC=RD, END";
          MOVS RD,R7 JMP @JUMP2         "MODE=2, PC=RD, MORE";
          MOVS RD,,MAR  READ  JMP @JUMP3 "MODE=3, RD=ADDR, MORE";
          D_SUB_S TWO,RD  JMP @JUMP4    "MODE=4,DEC RD,MORE";
          D_SUB_S TWO,RD,MAR READ JMP @JUMP5 "MODE=5,DEC RD,FTCH ADDR";
          MOVS R7,,MAR  READ  JMP @JUMP6 "MODE=6,FETCH INDEX,MORE";
          MOVS R7,,MAR  READ  JMP @JUMP7 "MODE=7,FETCH INDEX,MORE";
  @JUMP2: INC2 ,RD  NEXT                "AUTO INC, END";
  @JUMP3: INC2 ,RD                      "AUTO INC, MORE";
          MOVS DR0,R7  NEXT             "PC=(ADDR), END";
  @JUMP4: MOVS RD,R7  NEXT              "PC=ADDR, END";
  @JUMP5: CONT                          "WAIT FOR MEM, MORE";
          MOVS DR0,R7  NEXT             "PC=(ADDR), END";
  @JUMP6: INC2 ,R7                      "INC R7(VIZ PC+4), MORE";
          S_ADD_D DR0,R7  NEXT          "R7=PC+INDEX+4, END";
  @JUMP7: INC2 ,R7                      "INC R7(VIZ PC+4), MORE";
          S_ADD_D DR0,R7  READ          "FETCH ADDR, MORE";
          CONT                          "WAIT FOR FETCH";
          MOVS DR0,R7  NEXT             "R7=(PC+INDEX+4), END";


"RETURN FROM SUBROUTINE INSTRUCTION -CLASS RTS"

"RTS"
RTS:      MOVS RD,PC                    "RTS OP, PC=RD, MORE";
          MOVS R6,,MAR  READ            "SP TO MAR, MORE";
          INC2 ,R6                      "INC SP, MORE";
          MOVS DR0,RD  NEXT             "LINK REG= TOP OF STACK, END";


"CONDITION CODE INSTRUCTIONS - CLASS CC"
"CLN, CLK, CLV, CLC, CCC, SEN, SEZ,SEV, SEC, SCC"

CC:                                     "CLASS START ADDRESS FROM MAP"
                          "OPCODES 00 02 10 TO 00 02 37 ARE ILLEGAL"
          MOVS IMM,TEMP  PRESET=40Q     "TEMP=BAD CODE LIMIT, MORE";
          S_SUB_D FLDSEL(0,6),TEMP  USR "USR NEG IF CODE ILGL, MORE";
          CJP UM1 TADDR=ILGL            "JUMP IF CODE ILGL";
          MOVS PSW,TEMP                 "EXTRACT PSW";
          S_ADD_D FLDSEL(4,1),IMM
            TADDR=$+1  JMP INDEXED      "JMP FOR SET CC, MORE";
          NS_AND_D FLDSEL(0,4),TEMP
            STO_PSW  NEXT               "CLEAR CC, MSR=RESULT, MORE";
          S_OR_D FLDSEL(0,4),TEMP
            STO_PSW  NEXT               "SET CC, MSR=RESULT, END";
```

III.6 Class SWAB,BR and JSR

```
"SWAB INSTRUCTION - CLASS SWAB"

SWAB1:                                  "MODE EQ 0"
```

```
            MOVS RD,TEMP  JSR ROT8      "TEMP=RD, JUMP TO ROTATE BY 8";
            MOVS TEMP,RD  NEXT          "RESTORE TO RD, END";
  ROT8:
            MOVD ,TEMP  I6_9=0AH  I5_8=9  MSRZ  "ROTATE LEFT"
              PUSH UNCOND  PRESET=6     "LOAD COUNTER=6, MORE";
            MOVD ,TEMP  I6_9=0AH  I5_8=9  RFCT  "ROTATE, LOOP 6 TIMES";
            MOVS TEMP,TEMP@  I5_8=0EH   "TEMP@ IS DUMMY,SIGN EXT";
            MOVD ,TEMP@  MSR  RETURN    "SIGN EXTEND TO FIX CC, END";
SWAB2:                                  "MODE NE 0"
            S_ADD_D FLDSEL(3,3), IMM
              TADDR=DBASE  JSR INDEXED  "FETCH DST OPND,TADDR=DBASE+MODE";
            CONT                        "WAIT FOR MEM";
            MOVS DR0,TEMP  JSR ROT8     "TEMP=DR0, JUMP TO ROTATE BY 8";
            MOVS TEMP,DR0  WRITE  NEXT  "RESTORE TO DR0, END";


"BRANCH INSRUCTIONS"
"BR, BNE, BEQ, BGE, BLT  BGT, BLE"
"BRANCH INSTRUCTIONS - CLASS BR"

BR:         MOVS FLDSEL(0,8),TEMP  BSX  "TEMP=OFFSET(SIGN EXT),MORE";
            MOVD ,TEMP I5_8=9 I6_9=2    "OFFSET*2, MC UPDATE INHIBIT";
            S_ADD_D TEMP,R7  NEXT       "PC=PC+2*OFFSET, END";
BNE:        CJP MNE  TADDR=BR           "IF Z=0 THEN BRANCH";
            NEXT                        "ELSE END";
BEQ:        CJP MEQ  TADDR=BR           "IF Z=1 THEN BRANCH";
            NEXT                        "ELSE END";
BGE:        CJP MGE  TADDR=BR           "IF GE  THEN BRANCH";
            NEXT                        "ELSE END";
BLT:        CJP MLT  TADDR=BR           "IF LT  THEN BRANCH";
            NEXT                        "ELSE END";
BGT:        CJP MGT  TADDR=BR           "IF GT  THEN BRANCH";
            NEXT                        "ELSE END";
BLE:        CJP MLE  TADDR=BR           "IF LE  THEN BRANCH";
            NEXT                        "ELSE END";


"JSR INSTRUCTION - CLASS JSR@"

JSR@:                                   "CLASS START ADDR FROM MAP"
            S_ADD_D FLDSEL(3,3),IMM
              TADDR=$+1  JMP INDEXED    "DD MODE GIVES OFFSET, MORE";
  JSR0:     TRAP                        "MODE=0 IS ILLEGAL";
  JSR1:     MOVS RD,TEMP  JMP STAK0     "MODE=1, SAVE RD, MORE";
  JSR2:     MOVS RD,TEMP  JMP @JSR2     "MODE=2, SAVE RD, MORE";
  JSR3:     MOVS RD,,MAR  READ  JMP @JSR3 "MODE=3, READ (RD), MORE";
  JSR4:     D_SUB_S TWO,RD  JMP @JSR4   "MODE=4, DEC, MORE";
  JSR5:     D_SUB_S TWO,RD,MAR  READ  JMP @JSR5 "MODE=5,DEC,FTCH ADDR";
  JSR6:     MOVS R7,,MAR  READ  JMP @JSR6 "MODE=6, FETCH INDEX, MORE";
  JSR7:     MOVS R7,,MAR  READ  JMP @JSR7 "MODE=7, FETCH INDEX, MORE";
  @JSR2:    INC2 ,RD  JMP STAK0         "AUTO INC, MORE";
  @JSR3:    INC2 ,RD                    "AUTO INC, MORE";
            MOVS DR0,TEMP  JMP STAK0    "TEMP=DR0, MORE";
  @JSR4:    MOVS RD,TEMP  JMP STAK0     "TEMP=DR0, MORE";
  @JSR5:    CONT                        "WAIT FOR MEM";
            MOVS DR0,TEMP  JMP STAK0    "TEMP=OPND, MORE";
  @JSR6:    INC2 ,R7                    "INC R7, MORE";
            MOVS DR0,TEMP               "TEMP=INDEX, MORE";
            S_ADD_D R7,TEMP  JMP STAK0  "TEMP=ORIG PC+INDEX+4, MORE";
  @JSR7:    INC2 ,R7                    "INC R7, MORE";
            MOVS DR0,TEMP               "TEMP=INDEX, MORE";
```

```
          S_ADD_D R7,TEMP  READ        "TEMP=ORIG PC+INDEX+4, MORE";
          CONT                         "WAIT FOR MEM";
          MOVS DR0,TEMP  JMP STAK0     "TEMP=(ORIG PC+INDEX+4), MORE";
  STAK0:  MOVS RS,DR0                  "DR0=LINKAGE REG, MORE";
          D_SUB_S IMM,R6,MAR  WRITE  PRESET=2 "PUSH STACK";
          MOVS R7,RS                   "RS=PC";
          MOVS TEMP,R7  NEXT           "PC=SUBR ADDR";
```

III.7 Class SO1,SO2,SO3,SO4,SO5

```
"SINGLE OPERAND INSTRUCTIONS"
"CLR, COM, INC, DEC, NEG, ADC, SBC, TST, ROR, ROL, ASR, SXT, MARK,
 MFPI, MTPI"

"SINGLE OPERAND INSTRUCTIONS - CLASS SO1"

                                     " DST MODE EQ 0"
CLR1:     CLR ,RD   MSR  NEXT        "CLR OP, END";
COM1:     NOT_D ,RD   MSRCI  NEXT    "COM OP, END";
INC@1:    INC1 ,RD   MSRNOC  NEXT    "INC OP, END";
DEC1:     D_SUB_S ONE,RD  MSRNOC  NEXT "DEC OP, END";
NEG1:     NOT_D ,RD  I11_12=1  MSRCI  NEXT "NEG OP, END";
ADC1:     MOVD ,RD  I11_12=3  MSR NEXT "ADC OP, END";
SBC1:     MOVS ZERO,TEMP  I11_12=3  IO_5=MC "TEMP=CIN, MORE";
          D_SUB_S TEMP,RD  MSRCI  NEXT "DST-TEMP, END";
TST1:     MOVS RD  MSR  NEXT         "TST OP, END";

"SINGLE OPERAND INSTRUCTIONS - CLASS SO2"

SO2:                          "CLASS START ADDR FROM MAP,DST MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASE  JSR INDEXED  "FETCH DST OPND,TADDR=DBASE+MODE";
          S_ADD_D FLDSEL(6,3),IMM
            TADDR=$+1  JMP INDEXED    "JMP TO SO2 INSTR";
  CLR2:   CLR ,DR0  MSR  WRITE  NEXT  "CLR OP, END";
  COM2:   NOT_D ,DR0  MSRCI  WRITE  NEXT "COM OP, END";
  INC@2:  INC1 ,DR0  MSRNOC  WRITE  NEXT "INC OP, END";
  DEC2:   D_SUB_S ONE,DR0  MSRNOC  WRITE  NEXT "DEC OP, END";
  NEG2:   NOT_D  ,DR0  I11_12=1  MSRCI
            WRITE  NEXT               "NEG OP, END";
  ADC2:   MOVD ,DR0  I11_12=3
            MSR  WRITE NEXT           "ADC OP, END";
  SBC2:   MOVS ZERO,TEMP  I11_12=3  IO_5=MC
            JMP @SBC2                 "TEMP=CIN, MORE";
  TST2:   MOVS DR0  MSR  NEXT         "TST OP, END";
  @SBC2:  D_SUB_S TEMP,DR0  MSRCI  WRITE  NEXT "DST-TEMP, END";

"SINGLE OPERAND INSTRUCTIONS - CLASS SO3"

  SO3:                                "DST MODE EQ 0"
ROR3:     MOVD ,RD   ROR  JMP FXCC3   "ROR OP, END";
ROL3:     MOVD ,RD   ROL  JMP FXCC3   "ROL OP, END";
ASR3:     MOVS  RD   I6_9=9           "ASR OP, SHIFT N TO MC, MORE";
          MOVD ,RD  ROR     JMP FXCC3 "RIGHT SHIFT,MORE";
ASL3:     MOVD ,RD   ASL  JMP FXCC3   "ASL OP, END";
SXT3:     CJP MMI   TADDR=@SXT3       "SXT OP, MORE";
          CLR ,RD   MSR          NEXT "CLR IF NOT MMI, END";
```

```
  @SXT3:  NOT_S ONE,RD  I11_12=1  MSR  NEXT "RD=-1 IF MMI, END";
  FXCC3:  MOVS RD  USR   JMP FIXCC  "FIX COND CODES, EXCEPT OVR,MORE";


"SINGLE OPERAND INSTRUCTIONS - CLASS SO4"

SO4:                          "CLASS START ADDR FROM MAP,DST MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASE  JSR INDEXED  "FETCH DST OPND,TADDR=DBASE+MODE";
          S_ADD_D FLDSEL(6,3),IMM
            TADDR=$+1  JMP INDEXED    "JMP TO SO4 INSTR";
  ROR4:   MOVD ,DR0  ROR  WRITE  JMP FXCC4 "ROR OP, END";
  ROL4:   MOVD ,DR0  ROL  WRITE  JMP FXCC4 "ROL OP, END";
  ASR4:   MOVS  DR0  I6_9=9 JMP @ASR4 "ASR OP, SHIFT N TO MC, MORE";
  ASL4:   MOVD ,DR0  ASL  WRITE  JMP FXCC4 "ASL OP, END";
          TRAP;               "WRONG ADDR FOR SO5 GROUP(SEE BELOW)"
          TRAP;
          TRAP;
  SXT4:   CJP MMI   TADDR=@SXT4       "SXT OP, MORE";
          CLR ,DR0  MSR  WRITE  NEXT  "CLR IF NOT MMI, END";
  @SXT4:  NOT_S ONE,DR0 I11_12=1  MSRNOC
            NEXT WRITE                "-1 IF MMI, WRITE, END";
  @ASR4:  MOVD ,DR0  ROR  WRITE       "RIGHT SHIFT, END";
  FXCC4:  MOVS DR0  USR  JMP FIXCC "FIX COND CODES, EXCEPT OVR, MORE";


"SINGLE OPERAND INSTRUCTIONS - CLASS SO5"

MARK5:    MOVS FLDSEL(0,6),R6    ASL  "MARK OP, SP=2*NN, MORE";
          S_ADD_D R7,R6,MAR  READ     "SP=PC+2*NN, MORE";
          MOVS R5,R7                  "PC=R5, MORE";
          INC2 ,R6                    "AUTO INC SP, MORE";
          MOVS DR0,R5  NEXT           "RESTORE R5";


"DOUBLE OPERAND INSTRUCTIONS"

"MOV, CMP, BIT, BIC, BIS, ADD"


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO1"
                                      "SRC MODE EQ 0, DST MODE EQ 0"

  DO1:                                "MIRSRC IN RS, MIRDST IN RD"
MOV1:     MOVS RS,RD   MSRNOC NEXT    "MOV OP, END";
CMP1:     S_SUB_D RS,RD   TST  MSRCI  NEXT "CMP OP, END";
BIT1:     S_AND_D RS,RD  TST  MSRNOC  NEXT "BIT OP, END";
BIC1:     NS_AND_D RS,RD  MSRNOC NEXT "BIC OP, END";
BIS1:     S_OR_D RS,RD   MSRNOC  NEXT "BIS OP, END";
ADD1:     S_ADD_D RS,RD   MSR  NEXT   "ADD OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO2"

DO2:        "CLASS START ADDR FROM MAP, SRC MODE EQ 0, DST MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM     "FETCH DST OPND,TADDR=DBASE+MODE"
            TADDR=DBASE JSR INDEXED   "MIRSRC IN RS,MIRDST IN DR0";
          S_ADD_D FLDSEL(12,3),IMM
            TADDR=$  JMP INDEXED      "JMP TO DO2 INSTR";
  MOV2:   MOVS RS,DR0  MSRNOC NEXT  WRITE "MOV OP, END";
  CMP2:   S_SUB_D RS,DR0  TST  MSRCI  NEXT "CMP OP, END";
```

```
  BIT2:   S_AND_D RS,DR0  TST  MSRNOC  NEXT "BIT OP, END";
  BIC2:   NS_AND_D RS,DR0  MSRNOC NEXT WRITE "BIC OP, END";
  BIS2:   S_OR_D RS,DR0  MSRNOC  NEXT WRITE "BIS OP, END";
  ADD2:   S_ADD_D RS,DR0  MSR  NEXT  WRITE "ADD OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO3"

DO3:        "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE EQ 0"
          S_ADD_D FLDSEL(9,3),IMM
            TADDR=SBASE  JSR INDEXED  "FETCH SRC OPND,TADDR=DBASE+MODE";
          S_ADD_D FLDSEL(12,3),IMM    "MIRSRC IN DR0, MIRDST IN RD"
            TADDR=$  JMP INDEXED      "JMP TO DO3 INSTR";
  MOV3:   MOVS DR0,RD    MSRNOC NEXT  "MOV OP, END";
  CMP3:   S_SUB_D DR0,RD  TST  MSRCI  NEXT "CMP OP, END";
  BIT3:   S_AND_D DR0,RD  TST  MSRNOC  NEXT "BIT OP, END";
  BIC3:   NS_AND_D DR0,RD  MSRNOC  NEXT "BIC OP, END";
  BIS3:   S_OR_D DR0,RD  MSRNOC  NEXT "BIS OP, END";
  ADD3:   S_ADD_D DR0,RD  MSR  NEXT   "ADD OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO4"

DO4:        "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE NE 0"
          S_ADD_D FLDSEL(9,3),IMM
            TADDR=SBASE  JSR INDEXED  "FETCH SRC OPND,TADDR=DBASE+MODE";
          CONT                        "WAIT FOR OPND";
          MOVS DR0,TEMP               "STORE SRC OPND IN TEMP";
          S_ADD_D FLDSEL(3,3),IMM     "FETCH DST OPND,TADDR=DBASE+MODE"
            TADDR=DBASE JSR INDEXED   "MIRSRC IN TEMP, MIRDST IN DR0";
          S_ADD_D FLDSEL(12,3),IMM
            TADDR=$  JMP INDEXED      "JMP TO DO4 INSTR";
  MOV4:   MOVS TEMP,DR0  MSRNOC NEXT WRITE "MOV OP, END";
  CMP4:   S_SUB_D TEMP,DR0  TST  MSRCI  NEXT "CMP OP, END";
  BIT4:   S_AND_D TEMP,DR0  TST  MSRNOC  NEXT "BIT OP, END";
  BIC4:   NS_AND_D TEMP,DR0  MSRNOC  NEXT WRITE "BIC OP, END";
  BIS4:   S_OR_D TEMP,DR0  MSRNOC  NEXT  WRITE "BIS OP, END";
  ADD4:   S_ADD_D TEMP,DR0  MSR  NEXT WRITE "ADD OP, END";
```

III.8 Class EIS

```
"EXTENDED INSTRUCTION SET INSTRUCTIONS - CLASS EIS"
"MUL, DIV, ASH, ASHC, XOR, FADD, FSUB, FMUL, FDIV, SOB"

"MUL INSTRUCTION - DST(REG) AND SRC MULTIPLIED AND STORED IN REG,REG+1"

MUL1:                                 "SRC MODE EQ 0"
          MOVS RD,TEMP@               "TEMP@=SRC OPND";
  @MUL:                              "TEMP@=MULTIPLICAND, RS=MULTIPLIER"
          MOVS RS,QDST  LDCT  PRESET=14 "Q=MULTIPLIER, COUNTER=15";
          CLR ,TEMP                   "CLEAR TEMP, MORE";
          SMUL TEMP@,TEMP  I6 9=06H
            RPCT  TADDR=$    "2 COMP MULT,LINK S100 TO QI03, DECR LOOP";
          MULF TEMP@,TEMP I6 9=06H
            I11 12=2        "2 COMP MULT LAST CYCLE, LINK S100 TO QI03,
                             LINK CIN TO CX, MORE";
          "AT THIS STAGE PRODUCT(MS) IN TEMP
                         PRODUCT(LS) IN Q
                         MULTIPLICAND IN TEMP@
```

```
                        MULTIPLIER IN RS"
          MOVS TEMP,RS  JSR REGDST     "RS=PRODUCT(MS),
                                       REGDST PLACES (SS OR 1) IN DD FIELD"
               USR                     "STORE CC FOR MS, N CORRECT,
                                        UV=0, UC=0, MORE";
          MOVD ,RD,QIN  MSR            "STORE LS IN RD,
                                       STORE CC FOR LS IN MSR, MORE";
          "REST NEEDED TO FIX CONDITION CODES"
          CJP MPL TADDR=@MULA          "BR IF MS N=0, MORE";
          IO_5=17Q CEU=1               "SET UV, MORE";
  @MULA:                               "UV NOW CONTAINS N(LS)"
          CJP UGE TADDR=@MULB          "BR IF (N(MS) XOR N(LS))=0";
          IO_5=13Q CEU=1               "SET UC, MORE";
  @MULB:          "UC=(N(MS) XOR N(LS),VIZ REQUIRED C, MORE"
          IO_5=16Q CEU=1               "RESET UV, MORE";
          CJP MEQ  TADDR=@MULC         "JUMP IF LS Z=1, MORE";
          IO_5=10Q  CEU=1              "RESET UZ, MORE";
  @MULC:  IO_5=2 CEM=1 CEU=1 EC=1 NEXT "Z CORRECT, SWAP STATUS REG,
                                       END OF DIVIDE";
MUL2:                                  "SRC MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASE  JSR INDEXED   "FETCH SRC OPND, MORE";
          CONT                         "WAIT FOR OPND";
          MOVS DR0,TEMP@  JMP @MUL     "TEMP@=SRC OPND, MORE";


"DIV INSTRUCTION - DIVIDEND(MS) IN REG, DIVIDEND(LS) IN REG OR 1,
  SRC IS DIVISOR - PEFORMS TWOS COMPLEMENT DIVIDE"

DIV1:                                  "SRC MODE EQ 0"
          MOVS RD,DR0  MSR             "DR0=SRC OPND, SAVE DIVID SIGN";
  @DIV1:  MOVS DR0,QDST                "SAVE DR0";
          JSR REGDST                   "RD=DIVIDEND(LS), MORE";
          MOVD ,DR0,QIN                "RESTORE DR0";
  @DIV2:  MOVS DR0,TEMP IO_5=3 CEU=1   "TEMP=DIVISOR, RESET USR";
          MOVS RS,TEMP@                "TEMP@=DIVIDEND(MS)"
            CJP MEQ  TADDR=@DIVZ       "IF DIVISOR=0 THEN EXIT";
          CONV ,TEMP@  I11_12=2  USR   "TEMP@=S/M DIVIDEND(MS),
                                        LINK CIN TO CX, UPDATE USR";
          MOVD ,TEMP@  I6_9=2  I5_8=9  "TEMP@=2*TEMP@(MAG DVDMS)"
           CJP UVS TADDR=@QUOT         "ABORT IF OVERFLOW DURING CONV";
          CONV ,TEMP  I11_12=2   USR   "TEMP=S/M DIVISOR,
                                        LINK CIN TO CX, UPDATE USR";
          CJP UVS TADDR=@QUOT          "ABORT IF OVERFLOW DURING CONV";
          S_AND_D IMM,TEMP PRESET=77777Q "TEMP=MAG DVS";
          S_SUB_D TEMP@,TEMP  TST  USR "TEST DVDMS-DVS";
          CJP UHI  TADDR=@QUOT         "ABORT IF QUOTIENT TOO LARGE"
          MOVS RD,QDST                 "Q=RD(DIVIDEND(LS))";
          MOVD ,RS I5_8=0AH I6_9=4     "SHIFT DVDMS,DVDLS LEFT BEFORE
                                        STARTING DIVIDE";
          NMZD DR0,RS I6_9=0FH         "DOUBLE LENGTH NORM, SI00 TO QI03"
            PUSH UNCOND  PRESET=0DH    "PUSH LOC ON STACK,LOAD COUNTER";
          SDIV DR0,RS I11_12=2  I6_9=0FH "TWO COMP DIV, LINK CIN TO CX,
                        LINK SI00 TO QI03, SI03 TO QI00"
            RFCT                       "LOOP FOR 14 CYCLES";
          FDIV DR0,RS I11_12=2 I6_9=3 "DIVIDE CORR, LINK CIN TO CX,
                                        QI00=1";
          "QUOTIENT AND REMAINDER CORRECTION IS PERFORMED
           SIMILAR TO RHYNE(1971)"
          MOVD ,TEMP@,QIN   USR        "TEMP@=Q, UPDATE USR";
          MOVS DR0   MSR               "DR0=DVS, UPDATE MSR";
```

```
          MOVS RS,RD   CJP UMI          "RD=R"
           TADDR=@QNEG                  "BR IF Q NEG";
  @QPOS:  MOVS RD   CJP IO_5=17Q CCMUX=1
           TADDR=@ENDIV                 "BR IF R AND DVS SAME SIGN";
  @CORR1: D_SUB_S ONE,TEMP@             "TEMP@=Q-1";
          S_ADD_D DRO,RD  JMP @ENDIV    "RD=R+DVS";
  @QNEG:  MOVS RD   CJP IO_5=16Q CCMUX=1
           TADDR=@ENDIV                 "BR IF R AND DVS DIFF SIGN";
  @CORR2: S_ADD_D ONE,TEMP@             "TEMP@=Q+1";
          D_SUB_S DRO,RD                "RD=R-DVS";
  @ENDIV: MOVS TEMP@,RS   MSR  NEXT     "RS=QUOTIENT, UPDATE MSR"
                     "*******    END OF DIV OPERATION";
  @QUOT:  IO_5=3  CEU=1                 "RESET USR";
          IO_5=17Q  CEU=1               "SET UV AS QUOTIENT TOO LARGE";
  @DIVZ:  IO_5=13Q  CEU=1               "SET UC AS DIVIDE BY ZERO";
          IO_5=2Q CEU=1 CEM=1 EC=1 NEXT "SWAP USR,MSR,  END";
DIV2:                                   "SRC MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASE  JSR INDEXED    "FETCH SRC OPND, MORE";
          CONT                          "WAIT FOR SRC";
          MOVS DRO  MSR  JMP @DIV1      "DRO=SRC OPND, JUMP";


"ASH INSTRUCTION - REG SHIFTED ARITH NN PLACES LEFT OR RIGHT"
"ASH AND ASHC START AT SAME MAPPING ADDRESS"

ASH1:                                   "SRC MODE EQ 0"
          MOVS RD,TEMP@                 "TEMP@=SOURCE";
  @ASH:   MOVS TEMP@  USR               "UPDATE USR FOR SHIFT DIRN";
          CJP UMI  TADDR=@ASHR          "BR IF UN=1,VIZ RIGHT SHIFT";
          JMP @ASHL                     "LEFT SHIFT OP";
  @ASHR:                                "ASH, ASHC RIGHT SHIFT OPN"
          NOT_S TEMP@,IMM IO_5=5 CEU=1  "IMM=(MAG NN)-1, RESET USR"
           LDCT          "STORE MAG OF NO OF LOOPS IN COUNTER";
          S_ADD_D FLDSEL(9,1),IMM
           TADDR=$+1  JMP INDEXED       "BR IF ASHC";
          JMP @ASHRA                    ;
  @ASHCR:                               "ASHC RIGHT SHIFT OPERATION"
          JSR REGDST                    "RD=(RS OR 1)";
          MOVS RD,QDST                  "Q=REG(LS)";
          MOVS RS   I6_9=9              "SHIFT SIGN TO MC";
          MOVD ,RS I5_8=2 I6_9=0CH      "DOUBLE WORD ASR"
          RPCT TADDR=$-1                "LOOP FOR NN CYCLES";
          MOVS RS   USR JMP @ASHCC      "FIX COND CODES";
  @ASHRA:                               "ASH RIGHT SHIFT OPERATION"
          MOVS RS I6_9=9                "SHIFT SIGN TO MC";
          MOVD ,RS KOR                  "SHIFT RIGHT"
          RPCT TADDR=$-1                "LOOP FOR NN CYCLES";
          MOVS RS MSRNOC                "UPDATE MSR WITH MC RETAIN"
           NEXT                "********** END OF ASH (RIGHT SHIFT)";
  @ASHL:                                "ASH,ASHC LEFT SHIFT OPERATION"
          S_SUB_D TEMP@,IMM PRESET=1    "STORE (NN-1) IN LDCT"
           IO_5=3 CEU=1 LDCT            "RESET USR";
          S_ADD_D FLDSEL(9,1),IMM
           TADDR=$+1 JMP INDEXED        "BR IF ASHC";
          JMP @ASHLA;
  @ASHCL: JSR REGDST IO_5=3 CEU=1       "RD=(R OR 1), RESET USR";
          MOVS RD,QDST                  "Q=REG(LS)";
          MOVD ,RS I5_8=0BH I6_9=4      "DOUBLE WORD ASL"
           MSRNOC                       "MN=SIGN BEFORE SHIFT";
          MOVS RS CJP IO_5=17Q          "IN=SIGN AFTER SHIFT"
```

```
            CCMUX=1 TADDR=$+2          "BR IF (IN NXOR MN)=1";
            IO_5=17Q CEU=1             "SET UV";
            RPCT TADDR=$-3             "LOOP FOR NN CYCLES";
            MOVS RS IO_5=6 CEU=1       "UPDATE USR(V RETAIN)";
  @ASHCC:                              "FIX UZ FOR ASHC"
            "UN CORRECT, UV CORRECT, MC CORRECT"
            MOVD ,RD,QIN MSRNOC        "RD=LS WORD, MSR=LS STATUS";
            CJP MEQ TADDR=@ASHCD       "BR IF LS Z=1";
            IO_5=10Q CEU=1             "RESET UZ";
  @ASHCD:   IO_5=2 CEM=1 CEU=1 NEXT    "SWAP REG WITH MC RETAIN"
                                       "********* END OF ASHC OPERATION";
  @ASHLA:   MOVD ,RS ASL               "ASL"
             MSRNOC                    "MN=SIGN BEFORE SHIFT";
            MOVS RS CJP IO_5=17Q       "IN=SIGN AFTER SHIFT"
             CCMUX=1 TADDR=$+2         "BR IF (IN NXOR MN)=1";
            IO_5=17Q CEU=1             "SET UV";
            RPCT TADDR=$-3             "LOOP FOR NN CYCLES";
            MOVS RS IO_5=6 CEU=1       "UPDATE USR(V RETAIN)";
            IO_5=2 CEM=1 CEU=1 NEXT    "SWAP REG, MC RETAIN"
                        "**********"   END OF ASH (LEFT SHIFT)";
ASH2:                                  " SRC MODE NE 0"
            S_ADD_D FLDSEL(3,3),IMM
             TADDR=DBASE  JSR INDEXED  "FETCH SRC OPND,MORE";
            CONT                       "WAIT FOR OPND";
            MOVS DRO,TEMP@ JMP @ASH    "TEMP@=SRC OPND, MORE";


"XOR INSTRUCTION - DST=REG XOR DST"

XOR1:                                  "DST MODE EQ 0"
            S_XOR_D RS,RD  MSRNOC  NEXT "XOR OP, END";
XOR2:                                  "DST MODE NE 0"
            S_ADD_D FLDSEL(3,3),IMM
              TADDR=DBASE  JSR INDEXED "FETCH DST OPND, MORE";
            CONT                       "WAIT FOR OPND";
            S_XOR_D RS,DRO  MSRNOC  WRITE  NEXT "XOR OP, END";


"SOB INSTRUCTION - SUBTRACT ONE AND BRANCH IF NE 0"

SOB:        D_SUB_S ONE,RS  USR        "RS=RS-1, UPDATE USR";
            CJP UEQ  TADDR=FETCH      "IF RS ZERO THEN FETCH NEXT INSTR";
            MOVS FLDSEL(0,6),TEMP  ASL "TEMP=2*OFFSET, MORE";
            D_SUB_S TEMP,R7  NEXT      "R7=R7-2*OFFSET, END";



III.9  Class BRS,TR

"BRANCH INSTRUCTIONS"
"BPL, BMI, BHI, BLOS, BVC, BVS, BCC, BHIS, BCS, BLO"
"BRANCH INSTRUCTIONS - CLASS BRS"

  BRS:      MOVS FLDSEL(0,8),TEMP  BSX "TEMP=OFFSET(SIGN EXT),MORE";
            MOVD ,TEMP  15_8=9  16_9=2 "OFFSET*2,NO MC UPDATE,MORE";
            S_ADD_D TEMP,R7  NEXT      "PC=PC+2*OFFSET, END";
BPL:        CJP MPL  TADDR=BRS         "IF PLUS THEN BRANCH";
            NEXT                       "ELSE END";
BMI:        CJP MMI  TADDR=BRS         "IF MINUS THEN BRANCH";
            NEXT                       "ELSE END";
BHI:        CJP MHI  TADDR=BRS         "IF HIGHER THEN BRANCH";
```

```
          NEXT                              "ELSE END";
BLOS:     CJP MLOS  TADDR=BRS               "IF LOWER OR SAME THEN BRANCH";
          NEXT                              "ELSE END";
BVC:      CJP MVC   TADDR=BRS               "IF V CLEAR THEN BRANCH";
          NEXT                              "ELSE END";
BVS:      CJP MVS   TADDR=BRS               "IF V SET THEN BRANCH";
          NEXT                              "ELSE END";
BCC: BHIS: CJP MCC  TADDR=BRS               "IF C CLEAR THEN BRANCH";
          NEXT                              "ELSE END";
BCS: BLO: CJP MCS   TADDR=BRS               "IF C SET THEN BRANCH";
          NEXT                              "ELSE END";


"EMT AND TRAP INSTRUCTIONS"

EMT:      MOVS IMM,TEMP  PRESET=30Q  "VECTOR VIA 30Q";
          JMP GTVCTR                 "JUMP TO VECTOR";
TRAP@:    MOVS IMM,TEMP  PRESET=34Q  "VECTOR VIA 34Q";
          JMP GTVCTR                 "JUMP TO VECTOR";
```

III.10 Class SO6,SO7,SO8,SO9,SO10

```
"SINGLE OPERAND INSTRUCTIONS - BYTE"
"CLRB, COMB, INCB, NEGB, ADCB, TSTB"

"SINGLE OPERAND INSTRUCTIONS(BYTE) - CLASS SO6"

  SO6:                                      "DST MODE EQ 0"
CLR6:     CLR ,RD   MSR  NEXT               "CLRB OP, END";
COM6:     MOVS RD,TEMP  BSX                 "SIGN EXT, MORE";
          NOT_S TEMP,RD  STOB  MSRC1  NEXT "STOB OP, END";
INC@6:    MOVS RD,TEMP  BSX                 "SIGN EXT, MORE";
          S_ADD_D ONE,TEMP  MSRNOC          "INCB OP, MORE";
          MOVS TEMP,RD  STOB  NEXT          "WRITE LSB, END";
DEC6:     MOVS RD,TEMP  BSX                 "SIGN EXT, MORE";
          D_SUB_S ONE,TEMP  MSRNOC          "DECB OP, MORE";
          MOVS TEMP,RD  STOB  NEXT          "WRITE LSB, END";
NEG6:     MOVS RD,TEMP  BSX                 "SIGN EXT, MORE";
          NOT_S TEMP,RD  STOB  I11 12=1  MSRC1  NEXT "NEGB OP, END";
ADC6:     MOVS RD,TEMP  BSX                 "SIGN EXT, MORE";
          MOVS TEMP,RD  STOB  I11 12=3  MSR  NEXT "ADC OP";
SBC6:     MOVS RD,TEMP  BSX                 "SIGN EXT, MORE";
          MOVS ZERO,TEMP@  I11 12=3  I0_5=MC "TEMP@=CIN, MORE";
          D_SUB_S TEMP@,TEMP  MSRC1         "DST-TEMP@, MORE";
          MOVS TEMP,RD  STOB  NEXT          "WRITE LSB, END";
TST6:     MOVS RD,TEMP  BSX                 "TST OP,TEMP IS DUMMY";
@SO6:     MOVD ,TEMP  MSR  NEXT             "UPDATE MSR,END";


"SINGLE OPERAND INSTRUCTIONS(BYTE) - CLASS SO7"

SO7:                              "CLASS START ADDR FROM MAP,DST MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASEB  JSR INDEXED "FETCH DST OPND,TADDR=DBASE+MODE";
          CONT                          "WAIT FOR MEM, MORE";
          MOVD ,DRO  BSX                "SIGN EXT, MORE";
          S_ADD_D FLDSEL(6,3),IMM
            TADDR=$+1  JMP INDEXED      "JUMP TP SO7 INSTR, MORE";
  CLR7:   CLR ,DRO  MSR  WRITEB  NEXT "CLRB OP, END";
```

```
   COM7:   NOT_D ,DR0  MSRCI  WRITEB  NEXT "COMB OP, END";
   INC@7:  S_ADD_D ONE,DR0  MSRNOC  WRITEB  NEXT "INCB OP, END";
   DEC7:   D_SUB_S ONE,DR0  MSRNOC  WRITEB  NEXT "DECB OP, END";
   NEG7:   NOT_D  ,DR0  I11_12=1  MSRCI
             WRITEB  NEXT                "NEGB OP, END";
   ADC7:   MOVD ,DR0  I11_12=3
             MSR  WRITEB  NEXT           "ADCB OP, END";
   SBC7:   MOVS ZERO,TEMP  I11_12=3  IO_5=MC
             JMP @SBC7                   "TEMP=CIN, MORE";
   TST7:   MOVS DR0,TEMP  MSR  NEXT      "TSTB OP,TEMP IS DUMMY, END";
   @SBC7:  D_SUB_S TEMP,DR0  MSRCI  WRITEB  NEXT "DST-TEMP, END";


"SINGLE OPERAND INSTRUCTIONS (BYTE) - CLASS SO8"

   SO8:                              "MODE EQ 0"
 "CLEARS HIGH BYTE, ROTATES RIGHT, CORRECTS BIT7 IF NECESSARY"
 ROR8:     MOVS IMM,TEMP  IO_5=2 CEU=1 EC=1
             PRESET=0FFH               "TEMP=LOW BYTE MASK,USR=MSR";
           S_AND_D TEMP,RD   ROR  CEU=0
             CJP UCC  TADDR=@ROR8      "MASK LOW BYTE,ROTATE RIGHT,
                                       INHIBIT USR, JUMP IF UC CLEAR";
           S_OR_D  IMM,RD   PRESET=80H "IF UC SET CLEAR BIT7, MORE";
   @ROR8:  MOVD ,RD   BSX               "SIGN EXT, MORE";
   @SO8:   MOVD ,RD   USR  JMP FIXCC    "SAVE NZVC, EXCEPT OVR";
 ROL8:     MOVD ,RD   BSX             "SIGN EXTEND, MORE";
           MOVD ,RD   ROL  JMP @SO8   "ROTATE LEFT, MORE";
 ASR8:
 "CLEARS BITS 7-15, ROTATES RIGHT, CORRECTS BITS 6&7 IF NECESSARY"
           MOVS IMM,TEMP  PRESET=7FH  "TEMP=MASK FOR BIT0-6, MORE";
           MOVD ,RD  BSX              "SIGN EXTEND, MORE";
           MOVD ,RD  CEU=1            "SAVE SIGN IN USR";
           S_AND_D TEMP,RD   ROR  CEU=0
             CJP UPL  TADDR=@ASR8     "MASK BITS0-6, ROTATE RIGHT,
                                       INHIBIT USR, JUMP IF USR POS";
           S_OR_D IMM,RD   PRESET=0C0H "IF NEG , SET BITS 6&7, MORE";
   @ASR8:  MOVD ,RD   BSX  JMP @SO8   "SIGN EXT, SAVE EXCEPT OVR";
 ASL8:     MOVD ,RD   BSX             "SIGN EXTEND OPND, MORE";
           MOVD ,RD   ASL             "SHIFT LEFT, SAVE, MORE";
           MOVS RD   USR  JMP FIXCC   "FIX CC, EXCEPT OVR";


 "SINGLE OPERAND INSTRUCTIONS(BYTE) - CLASS SO9"

   SO9:                              "DST MODE NE 0"
 "CLEARS HIGH BYTE, ROTATES RIGHT, CORRECTS BIT7 IF NECESSARY"
 ROR9:     S_ADD_D FLDSEL(3,3),IMM
             TADDR=DBASEB  JSR INDEXED "FETCH DST OPND, MORE";
           MOVS IMM,TEMP  IO_5=2
            CEU=1 EC=1    PRESET=0FFH  "TEMP=LOW BYTE MASK,USR=MSR";
           S_AND_D TEMP,DR0  ROR  CEU=0
             CJP UCC  TADDR=@ROR9      "MASK LOW BYTE,ROTATE RIGHT,
                                       INHIBIT USR, JUMP IF UC CLEAR";
           S_OR_D  IMM,DR0  PRESET=80H "IF UC SET CLEAR BIT7, MORE";
   @ROR9:  MOVD ,DR0  BSX   WRITEB    "SIGN EXT, MORE";
   @SO9:   MOVD ,DR0  USR  JMP FIXCC  "SAVE NZVC EXCEPT OVR";
 ROL9:     S_ADD_D FLDSEL(3,3),IMM
             TADDR=DBASEB  JSR INDEXED "FETCH DST OPND, MORE";
           CONT                       "WAIT FOR MEM, MORE";
           MOVD ,DR0  BSX             "SIGN EXTEND, MORE";
           MOVD ,DR0  ROL  WRITEB     "ROTATE LEFT, MORE";
```

```
          MOVS DRO  USR      JMP FIXCC "FIX CC EXCEPT OVR";
ASR9:
"CLEARS BITS 7-15, ROTATES RIGHT, CORRECTS BITS 6&7 IF NECESSARY"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASEB  JSR INDEXED "FETCH DST OPND,MORE";
          MOVS IMM,TEMP  PRESET=7FH   "TEMP=MASK FOR BIT0-6, MORE";
          MOVD ,DRO  BSX              "SIGN EXTEND, MORE";
          MOVD ,DRO  CEU=1            "SAVE SIGN IN USR";
          S_AND_D TEMP,DRO  ROR  CEU=0
            CJP UPL  TADDR=@ASR9      "MASK BITS0-6, ROTATE RIGHT,
                                       INHIBIT USR, JUMP IF USR POS";
          S_OR_D IMM,DRO  PRESET=0C0H "IF NEG, SET BITS 6&7, MORE";
  @ASR9:  MOVD ,DRO  BSX  WRITEB      "SIGN EXT, MORE"
           JMP @SO9                   ;
ASL9:     S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASEB  JSR INDEXED "FETCH DST OPND, MORE";
          CONT                        "WAIT FOR MEM, MORE";
          MOVD ,DRO  BSX              "SIGN EXTEND OPND, MORE";
          MOVD ,DRO  ASL  WRITEB      "SHIFT LEFT, SAVE, MORE";
          MOVS DRO  USR      JMP FIXCC "FIX CC, EXCEPT OVR";


"SINGLE OPERAND INSTRUCTIONS - CLASS SO10"

  SO10:

"MTPS MOVES SRC TO PSW"

MTPS1:                                "MODE EQ 0"
          MOVD ,RD       BSX          "SIGN EXT, MORE";
          MOVS RD,DUMP STO_PSW  NEXT "PSW=SRC,END";
MTPS2:                                "MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASEB  JSR INDEXED "FETCH SRC OPND, MORE";
          CONT                        "WAIT FOR SRC, MORE";
          MOVD ,DRO      BSX          "SIGN EXT,MORE";
          MOVS DRO,DUMP  STO_PSW  NEXT "PSW=SRC,END";


"MFPS MOVES CONTENTS OF PSW TO DST"

MFPS1:                                "MODE EQ 0"
          MOVS PSW,TEMP               "EXTRACT PSW, STORE IN TEMP";
          MOVS TEMP,RD  BSX           "SIGN EXT DST, STORE, MORE";
  @MFPS1: MOVD ,RD  MSRNOC  NEXT      "SAVE NZV, END";
MFPS2:                                "MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM
            TADDR=DBASEB  JSR INDEXED "FETCH DST OPND, MORE";
          MOVS PSW,TEMP               "EXTRACT PSW, STORE IN TEMP";
          MOVS TEMP,DRO  BSX
            WRITEB                    "SIGN EXT DST, STORE, END";
  @MFPS2: MOVD ,DRO  MSRNOC  NEXT     "SAVE NZV, END";
```

III.11 Class DO5,DO6,DO7,DO8,DO9

```
"DOUBLE OPERAND INSTRUCTIONS(BYTE)"
"MOVB, CMPB, BITB, BICB, BISB"

"DOUBLE OPERAND INSTRUCTIONS - CLASS DO5"
```

```
D05:                                    "SRC MODE EQ 0, DST MODE EQ 0"
"DO NOT MODIFY HIGH BYTE OF RD EXCEPT FOR MOV"
          MOVS RS,TEMP  BSX             "TEMP=SRC, SIGN EXT, MORE";
          MOVS RD,TEMP@  BSX            "TEMP@=DST, SIGN EXT, MORE";
          S_ADD_D FLDSEL(12,3),IMM      "MIRSRC IN RS, MIRDST IN RD"
            TADDR=$  JMP INDEXED        "JUMP TO D05 INSTR";
  MOV5:   MOVS TEMP,RD  BSX  JMP @MFPS1 "MOV OP, MORE";
  CMP5:   S_SUB_D TEMP,TEMP@   TST  MSRCI  NEXT "CMP OP, END";
  BIT5:   S_AND_D TEMP,TEMP@  TST  MSRNOC  NEXT "BIT OP, END";
  BIC5:   NS_AND_D RS,RD  STOB  MSRNOC NEXT "BIC OP, END";
  BIS5:   S_OR_D RS,RD  STOB  MSRNOC  NEXT "BIS OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS D06"

D06:        "CLASS START ADDR FROM MAP, SRC MODE EQ 0, DST MODE NE 0"
          S_ADD_D FLDSEL(3,3),IMM       "FETCH DST OPND,TADDR=DBASE+MODE"
            TADDR=DBASEB  JSR INDEXED "MIRSRC IN RS,MIRDST IN DR0";
          MOVS RS,TEMP  BSX             "TEMP=MIRSRC, SIGN EXT, MORE";
          MOVD ,DR0  BSX                "SIGN EXTEND DST, MORE";
          S_ADD_D FLDSEL(12,3),IMM
            TADDR=$  JMP INDEXED        "JMP TO D06 INSTR";
  MOV6:   MOVS TEMP,DR0  MSRNOC NEXT  WRITEB "MOV OP, END";
  CMP6:   S_SUB_D TEMP,DR0  TST  MSRCI  NEXT "CMP OP, END";
  BIT6:   S_AND_D TEMP,DR0  TST  MSRNOC  NEXT "BIT OP, END";
  BIC6:   NS_AND_D TEMP,DR0  MSRNOC NEXT WRITEB "BIC OP, END";
  BIS6:   S_OR_D TEMP,DR0  MSRNOC  NEXT WRITEB "BIS OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS D07"
"DO NOT MODIFY HIGH BYTE OF RD EXCEPT FOR MOVB"

D07:        "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE EQ 0"
          S_ADD_D FLDSEL(9,3),IMM
            TADDR=SBASEB JSR INDEXED  "FETCH SRC OPND,TADDR=DBASE+MODE";
          MOVS RD,TEMP  BSX             "TEMP=SIGN EXTEND DST, MORE";
          MOVD ,DR0  BSX                "SIGN EXTEND SRC, MORE";
          S_ADD_D FLDSEL(12,3),IMM      "MIRSRC IN DR0, MIRDST IN RD"
            TADDR=$  JMP INDEXED        "JMP TO D07 INSTR";
  MOV7:   MOVS DR0,RD  BSX JMP @MFPS1 "MOV OP, SIGN EXT, MORE";
  CMP7:   S_SUB_D DR0,TEMP  TST  MSRCI  NEXT "CMP OP, END";
  BIT7:   S_AND_D DR0,TEMP  TST  MSRNOC  NEXT "BIT OP, END";
  BIC7:   NS_AND_D DR0,RD  STOB  MSRNOC  NEXT "BIC OP, END";
  BIS7:   S_OR_D DR0,RD  STOB  MSRNOC  NEXT "BIS OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS D08"

D08:        "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE NE0"
          S_ADD_D FLDSEL(9,3),IMM
            TADDR=SBASEB JSR INDEXED  "FETCH SRC OPND,TADDR=DBASE+MODE";
          CONT                          "WAIT FOR OPND";
          MOVS DR0,TEMP  BSX            "TEMP=SRC OPND, SIGN EXTEND";
          S_ADD_D FLDSEL(3,3),IMM       "FETCH DST OPND,TADDR=DBASE+MODE"
            TADDR=DBASEB  JSR INDEXED "MIRSRC IN TEMP, MIRDST IN DR0";
          CONT                          "WAIT FOR DST, MORE";
          MOVD ,DR0  BSX                "SIGN EXTEND DST, MORE";
          S_ADD_D FLDSEL(12,3),IMM
            TADDR=$  JMP INDEXED        "JMP TO D08 INSTR";
  MOV8:   MOVS TEMP,DR0  MSRNOC NEXT WRITEB "MOV OP, MORE";
  CMP8:   S_SUB_D TEMP,DR0  TST  MSRCI  NEXT "CMP OP, MORE";
```

```
BIT8:   S_AND_D TEMP,DRO  TST  MSRNOC  NEXT "BIT OP, MORE";
BIC8:   NS_AND_D TEMP,DRO  MSRNOC  NEXT WRITEB "BIC OP, END";
BIS8:   S_OR_D TEMP,DRO  MSRNOC  NEXT  WRITEB "BIS OP, MORE";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO9"

SUB1:                                   "SRC MODE EQ 0, DST MODE EQ 0"
                                        "MIRSRC IN RS, MIRDST IN RD"
        D_SUB_S RS,RD  MSRCI  NEXT  "SUB OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO10"

SUB2:       "CLASS START ADDR FROM MAP, SRC MODE EQ 0, DST MODE NE 0"
        S_ADD_D FLDSEL(3,3),IMM     "FETCH DST OPND,TADDR=DBASE+MODE"
          TADDR=DBASE JSR INDEXED   "MIRSRC IN RS,MIRDST IN DRO";
        CONT                        "WAIT FOR DST, MORE";
        D_SUB_S RS,DRO  MSRCI  NEXT  WRITE "SUB OP, MORE";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO11"

SUB3:       "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE EQ 0"
        S_ADD_D FLDSEL(9,3),IMM
          TADDR=SBASE  JSR INDEXED  "FETCH SRC OPND,TADDR=DBASE+MODE";
        CONT                        "WAIT FOR SRC, MORE";
        D_SUB_S DRO,RD  MSRCI  NEXT "SUB OP, END";


"DOUBLE OPERAND INSTRUCTIONS - CLASS DO12"

SUB4:       "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE NE0"
        S_ADD_D FLDSEL(9,3),IMM
          TADDR=SBASE  JSR INDEXED  "FETCH SRC OPND,TADDR=DBASE+MODE";
        CONT                        "WAIT FOR MEM, MORE";
        MOVS DRO,TEMP               "TEMP=MIRSRC, MORE";
        S_ADD_D FLDSEL(3,3),IMM     "FETCH DST OPND,TADDR=DBASE+MODE"
          TADDR=DBASE   JSR INDEXED "MIRSRC IN TEMP, MIRDST IN DRO";
        CONT                        "WAIT FOR MEM, MORE";
        D_SUB_S TEMP,DRO  MSRCI  NEXT  WRITE "SUB OP, END";


III.12 Indirect mapping table

"THE FOLLOWING IS A TEMPORARY JUMP TABLE TO AVOID
 THE NECESSITY OF REBURNING PROMS FOR EACH CORRECTION"

        ORG 1140Q;
JILGL:  JMP ILGL;
JM1:    JMP M1;
JJUMP:  JMP JUMP;
JRTS:   JMP RTS;
JCC:    JMP CC;
JSWB1:  JMP SWAB1;
JSWB2:  JMP SWAB2;
JBR:    JMP BR;
JBNE:   JMP BNE;
JBEQ:   JMP BEQ;
JBGE:   JMP BGE;
```

```
JBLT:   JMP BLT;
JBGT:   JMP BGT;
JBLE:   JMP BLE;
JJSR@:  JMP JSR@;
JCLR1:  JMP CLR1;
JCOM1:  JMP COM1;
JINC1:  JMP INC@1;
JDEC1:  JMP DEC1;
JNEG1:  JMP NEG1;
JADC1:  JMP ADC1;
JSBC1:  JMP SBC1;
JTST1:  JMP TST1;
JSO2:   JMP SO2;
JROR3:  JMP ROR3;
JROL3:  JMP ROL3;
JASR3:  JMP ASR3;
JASL3:  JMP ASL3;
JSXT3:  JMP SXT3;
JSO4:   JMP SO4;
JMARK:  JMP MARK5;
JMOV1:  JMP MOV1;
JCMP1:  JMP CMP1;
JBIT1:  JMP BIT1;
JBIC1:  JMP BIC1;
JBIS1:  JMP BIS1;
JADD1:  JMP ADD1;
JDO2:   JMP DO2;
JDO3:   JMP DO3;
JDO4:   JMP DO4;
JMUL1:  JMP MUL1;
JMUL2:  JMP MUL2;
JDIV1:  JMP DIV1;
JDIV2:  JMP DIV2;
JASH1:  JMP ASH1;
JASH2:  JMP ASH2;
JXOR1:  JMP XOR1;
JXOR2:  JMP XOR2;
JSOB:   JMP SOB;
JBPL:   JMP BPL;
JBMI:   JMP BMI;
JBHI:   JMP BHI;
JBLOS:  JMP BLOS;
JBVC:   JMP BVC;
JBVS:   JMP BVS;
JBCC:   JMP BCC;
JBCS:   JMP BCS;
JEMT:   JMP EMT;
JTRAP:  JMP TRAP@;
JCLR6:  JMP CLR6;
JCOM6:  JMP COM6;
JINC6:  JMP INC@6;
JDEC6:  JMP DEC6;
JNEG6:  JMP NEG6;
JADC6:  JMP ADC6;
JSBC6:  JMP SBC6;
JTST6:  JMP TST6;
JSO7:   JMP SO7;
JROR8:  JMP ROR8;
JROL8:  JMP ROL8;
JASR8:  JMP ASR8;
JASL8:  JMP ASL8;
```

```
JROR9:    JMP ROR9;
JROL9:    JMP ROL9;
JASR9:    JMP ASR9;
JASL9:    JMP ASL9;
JMTP1:    JMP MTPS1;
JMTP2:    JMP MTPS2;
JMFP1:    JMP MFPS1;
JMFP2:    JMP MFPS2;
JDO5:     JMP DO5;
JDO6:     JMP DO6;
JDO7:     JMP DO7;
JDO8:     JMP DO8;
JSUB1:    JMP SUB1;
JSUB2:    JMP SUB2;
JSUB3:    JMP SUB3;
JSUB4:    JMP SUB4;
JSPC:     JMP SPC;
```

III.13 Special macro-instructions

```
SPC:
          "START OF SPECIAL OPERATION INSTRUCTIONS"
          "DECODES 1700XX TO 1777XX ONLY"
          "SPECIALS ARE IN 8 GROUPS - SPC0..SPC7"
          "(CORRESPONDS TO 170XXX..177XXX)"
          S_ADD_D FLDSEL(9,3),IMM
           TADDR=$+1  JMP INDEXED      "JUMP TO SPC GROUP";
          S_ADD_D FLDSEL(6,3),IMM
           TADDR=SPC0  JMP INDEXED     "JUMP WITHIN SPC0(1 OF 8)";
          S_ADD_D FLDSEL(8,1),IMM
           TADDR=SPC1   JMP INDEXED    "JUMP WITHIN SPC1(1 OF 2)";
          S_ADD_D FLDSEL(8,1),IMM
           TADDR=SPC2   JMP INDEXED    "JUMP WITHIN SPC2(1 OF 2)";
          JMP ILGL                     "SPC3 NOT USED";
          JMP ILGL                     "SPC4 NOT USED";
          JMP ILGL                     "SPC5 NOT USED";
          JMP ILGL                     "SPC6 NOT USED";
          JMP ILGL                     "SPC7 NOT USED";
SPC0:     "MFCPU,MTCPU,MFCAC,MTCAC,MFIOP,MTHST,MTREF"
          "TEMP USED TO MANIPULATE ADDRESSING MODE"
          S_ADD_D FLDSEL(3,3),IMM
           TADDR=MFCPU  JMP INDEXED    "MFCPU ADDRESSING";
          S_ADD_D FLDSEL(3,3),IMM
           TADDR=MTCPU  JMP INDEXED    "MTCPU ADDRESSING";
          S_ADD_D FLDSEL(3,3),IMM
           TADDR=MFCAC  JMP INDEXED    "MFCAC ADDRESSING";
          S_ADD_D FLDSEL(3,3),IMM
           TADDR=MTCAC  JMP INDEXED    "MTCAC ADDRESSING";
          S_ADD_D FLDSEL(3,3),IMM
           TADDR=MFIOP  JMP INDEXED    "MFIOP ADDRESSING";
          S_ADD_D FLDSEL(3,3),IMM
           TADDR=MTIOP  JMP INDEXED    "MTIOP ADDRESSING";
          JMP MTHST                    "MTHST INTERPRETING";
          DST=26Q  NEXT                "MTREF INTERPRETING";
MFCPU:    "MOVES FROM CPU TO DR2,DR1,DR0 (NOTE ORDER)"
          "ADDRESSING RESOLVED FIRST"
          MOVS R2,DR2  JMP @MFCP0      "R2=DR2(MODE=0)";
          JMP @MFCP1   CLR ,TEMP       "MODE=1";
          JMP @MFCP2   CLR ,TEMP       "MODE=2";
```

```
            JMP ILGL                        "MODE=3 IS ILLEGAL";
            JMP @MFCP4   CLR ,TEMP          "MODE=4";
            JMP ILGL                        "MODE=5 IS ILLEGAL";
            JMP ILGL                        "MODE=6 IS ILLEGAL";
            JMP ILGL                        "MODE=7 IS ILLEGAL";
 @MFCP0:    MOVS R1,DR1                     "DR1=R1";
            MOVS R0,DR0  JMP FCHSPC         "DR0=R0, END";
 @MFCP4:    "ENTRY POINT FOR MODE=4, SUBTRACT 6 FROM POINTER"
            D_SUB_S IMM,RD  PRESET=6        "ADJUST RD (REG ADDR)";
            JMP @MFCP1                      "EXECUTE INSTR";
 @MFCP2:    "ENTRY POINT FOR MODE=2, TEMP=2 FOR THIS MODE"
            MOVS TWO,TEMP  JMP @MFCP        "TEMP FOR POINTER CORRN";
 @MFCP1:    "ENTRY POINT FOR MODE=1"
            MOVS IMM,TEMP PRESET=-4         "ADJ ADDR";
 @MFCP:     MOVS RD,,MAR  READ              "GET EXP";
            INC2 ,RD,MAR                    "ADJ MAR";
            MOVS DR0,TEMP@  READ            "TEMP@=EXP, GET MSM";
            INC2 ,RD,MAR                    "ADJ MAR";
            MOVS DR0,R10  READ              "R10=MSM, GET LSM";
            S_ADD_D TEMP,RD                 "ADJ ADDR, DR0=LSM";
            MOVS R10,DR1                    "DR1=MSM";
            MOVS TEMP@,DR2  JMP FCHSPC      "DR2=EXP, END";
 MTCPU:     "MOVES TO CPU FROM DR0,DR1,DR2(NOTE ORDER)"
            "ADDRESSING RESOLVED FIRST"
            MOVS DR0,R0  JMP @MTCP0         "MODE=0";
            JMP @MTCP1   CLR ,TEMP          "MODE=1";
            JMP @MTCP2   CLR ,TEMP          "MODE=2";
            JMP ILGL                        "MODE=3 IS ILLEGAL";
            JMP @MTCP4   CLR ,TEMP          "MODE=4";
            JMP ILGL                        "MODE=5 IS ILLEGAL";
            JMP ILGL                        "MODE=6 IS ILLEGAL";
            JMP ILGL                        "MODE=7 IS ILLEGAL";
 @MTCP0:    MOVS DR1,R1                     "R1=DR1";
            MOVS DR2,R2  NEXT               "R2=DR2, END";
 @MTCP4:    D_SUB_S IMM,RD  PRESET=6        "ADJUST POINTER FOR MODE=4";
            JMP @MTCP1;
 @MTCP2:    MOVS IMM,TEMP  PRESET=6         "CORRN FOR ADDR POINTER";
 @MTCP1:    "EP FOR MODE=1(TEMP PRESET TO 0)"
            S_ADD_D IMM,RD,MAR  PRESET=4  WRITE "ADJ PTR,WRITE LSM";
            MOVS DR1,DR0                    "DR0=MSM";
            D_SUB_S TWO,RD,MAR  WRITE       "ADJ PTR, WRITE MSM";
            MOVS DR2,DR0                    "DR0=EXP";
            D_SUB_S TWO,RD,MAR  WRITE       "ADJ PTR, WRITE EXP";
            S_ADD_D TEMP,RD  NEXT           "CORR PTR, END";
 MFCAC:     "MOVE FROM CACHE TO DR2,DR1,DR0"
            "ADDRESSING RESOLVED FIRST"
            JMP ILGL                        "MODE=0 IS ILLEGAL";
            MOVS RD,,MAR READC JMP DELAY    "MODE=1";
            MOVS RD,,MAR READC JMP @MFCA2   "MODE=2";
            JMP ILGL                        "MODE=3 IS ILLEGAL";
            JMP @MFCA4                      "MODE=4";
            JMP ILGL                        "MODE=5 IS ILLEGAL";
            JMP ILGL                        "MODE=6 IS ILLEGAL";
            JMP ILGL                        "MODE=7 IS ILLEGAL";
 @MFCA2:    INC1 ,RD  JMP FCHSPC            "ADJUST POINTER, END";
 @MFCA4:    D_SUB_S ONE,RD,MAR  READC       "ADJUST POINTER, READ";
 DELAY:     JMP FCHSPC                      "WAIT FOR READ, END";
 MTCAC:     "MOVE TO CACHE FROM DR0,DR1,DR2"
            "ADDRESSING RESOLVED FIRST"
            JMP ILGL                        "MODE=0 IS ILLEGAL";
            MOVS RD,,MAR  WRITEC  NEXT      "MODE=1";
```

```
          MOVS RD,,MAR WRITEC JMP @MTCA2 "MODE=2";
          JMP ILGL                       "MODE=3 IS ILLEGAL";
          JMP @MTCA4                     "MODE=4";
          JMP ILGL                       "MODE=5 IS ILLEGAL";
          JMP ILGL                       "MODE=6 IS ILLEGAL";
          JMP ILGL                       "MODE=7 IS ILLEGAL";
 @MTCA2:  INC1 ,RD  NEXT                 "ADJUST POINTER, END";
 @MTCA4:  D_SUB_S ONE,RD,MAR WRITEC NEXT "ADJ PNTER,WRITE";
MFIOP:    "MOVES FROM 40BIT IO PAGE TO DR0,DR1,DR2"
          "ADDRESSING RESOLVED FIRST"
          JMP ILGL                       "MODE=0";
          JMP @MFIO1                     "MODE=1";
          JMP ILGL                       "MODE=2";
          JMP ILGL                       "MODE=3";
          JMP ILGL                       "MODE=4";
          JMP ILGL                       "MODE=5";
          JMP ILGL                       "MODE=6";
          JMP ILGL                       "MODE=7";
 @MFIO1:  MOVS RD,,MAR  READ             "FETCH DST OPND INTO DR0,1,2";
          JMP FCHSPC                     "WAIT THEN FETCH NO INTRPT";
MTIOP:    "MOVES TO 40BIT IO PAGE TO DR0,DR1,DR2"
          "ADDRESSING RESOLVED FIRST"
          JMP ILGL                       "MODE=0";
          JMP @MTIO1                     "MODE=1";
          JMP ILGL                       "MODE=2";
          JMP ILGL                       "MODE=3";
          JMP ILGL                       "MODE=4";
          JMP ILGL                       "MODE=5";
          JMP ILGL                       "MODE=6";
          JMP ILGL                       "MODE=7";
 @MTIO1:  MOVS RD,,MAR  WRITE  NEXT      "WRITE DR0,1,2 TO IO PAGE";
MTHST:    "MOVES DST TO OPREG"
          S_AND_D FLDSEL(3,3),IMM  TADDR=7  USR "UZ=1 IF MODE=0";
          CJP UNE TADDR=@MTHST1          "JMP IF MODE NE 0";
          MOVS RD,OPREG  MSR  NEXT       "OPREG=DST(MODE=0)";
 @MTHST1: S_ADD_D FLDSEL(3,3),IMM
           TADDR=DBASE  JSR INDEXED      "FETCH DST OPND";
          CONT                           "WAIT FOR MEMORY";
          MOVS DR0,OPREG  MSR  NEXT      "OPREG=DST(MODE NE 0)";
SPC1:     "INTERPRET BDGTR,BDLER"
          JMP @SPC1                      "ITS BDGTR";
          CJP CCMUX=6 TADDR=BR           "BR IF COND TRUE(BDLER)";
          NEXT                           "OTHERWISE END";
 @SPC1:   CJP CCMUX=7  TADDR=BR          "BR IF COND TRUE(BDGTR)";
          NEXT                           "OTHERWISE END";
SPC2:     "INTERPRET BLDGTR,BLDLER"
          JMP @SPC2                      "ITS BLDGTR";
          CJP CCMUX=6 IO_5=45Q TADDR=BR "BR IF COND TRUE(BLDLER)";
          NEXT                           "OTHERWISE END";
 @SPC2:   CJP CCMUX=7 IO_5=45Q TADDR=BR "BR IF COND TRUE(BLDGTR)";
          NEXT                           "OTHERWISE END";
END;
```

APPENDIX IV

OPERATING PROCEDURES ON THE IBM370

As described in reference 2, the formatted microcode is generated in a two pass process. The first pass generates the microcode in an intermediate object code form. This is accomplished by the foreground job 'ARO.LIB.CLIST(ASMDEV)'. In practice this task takes two minutes of CPU time so the BATCHTSO command is used to submit this portion of the job. This job processes the source code found in 'ARO.CPROC.DATA' and generates a listing file 'ARO.CPROC.MLST' and an intermediate object code file 'ARO.CPROC.MOBJ'.

The second pass reformats 'ARO.CPROC.MOBJ' into the load module 'ARO.CPROC.MFMT'. This is accomplished by the foreground job 'ARO.LIB.CLIST(FMTRUN)'.

The next stage is to extract the addresses of the left justified labels (see Sections 5.3 and 5.4) from the listing file to generate the contents of the mapping PROMs. Additional information on the PDP-11 instruction code is required at this stage so that the left justified addresses can be associated with particular groups of instruction codes. The foreground job 'ARO.LIB.CLIST(CPROC)' takes the listing file 'ARO.CPROC.MLST' and PDP-11 code information in 'ARO.JMAP.DATA' and generates the mapping PROM contents in 'ARO.PROM.DAT'. The latter file contains a list of the PROM addresses and contents and mnemonics of the left justified labels. Due to the design of the Control Processor, the PROM addresses are mapped from the PDP-11 op-code as follows:- the most significant ten bits of the op-code are directly mapped to the most significant ten bits of the PROM address and the least significant bit of the PROM address is generated from NOT(BIT3 OR BIT4 OR BIT5) of the op-code (viz. detects mode=0 in the destination field). Thus destination mode=0 is mapped to odd PROM addresses and destination mode=1 through to mode=7 are mapped to even PROM addresses.

The listing file 'ARO.CPROC.MLST' cannot be generated with octal microcode addresses as is the PDP-11 standard. The foreground job 'ARO.LIB.CLIST(CPOCT)' reads the listing file and produces 'ARO.CPOCT.MLST' which is in the desired format.

The final stage is to generate the magnetic tape volume ARO1 which is in RSX FILES-11 format containing the files (7,75)MICROCODE.DAT and (7,75)PROM.DAT. This is accomplished by submitting the background job 'ARO.LIB.CNTL(RSXDJH)'. When the data is transferred to the Jindalee Auxiliary Computer, PROM.DAT is converted into the form required by the PROM programmer by the program (7,75)AROPRM.FTN.

IV.1 Usage summary

The source for the microcode is stored in 'ARO.CPROC.DATA'. To recompile the microcode enter:

```
BATCH DSN(LIB.CLIST(ASMDEV)) CLASS(A) MSGCLASS(T) JOBCHAR(Q)
```

To display the results of the compilation, reformat the microcode, generate the mapping PROM contents, convert the addresses to octal, and print the listing file, enter:

```
EX LIB(GENUC)
```

To transfer the formatted microcode and PROM contents to RSX FILES-11 tape enter:

SUBMIT LIB(RSXDJH)

It is essential to check that the following data sets are in the catalog:
'ARO.LIB.LOAD','ARO.LIB.CLIST','ARO.LIB.CNTL',
'ARO.JMAP.DATA','ARO.CPROC.DATA','ARO.CPROC.CMND'.

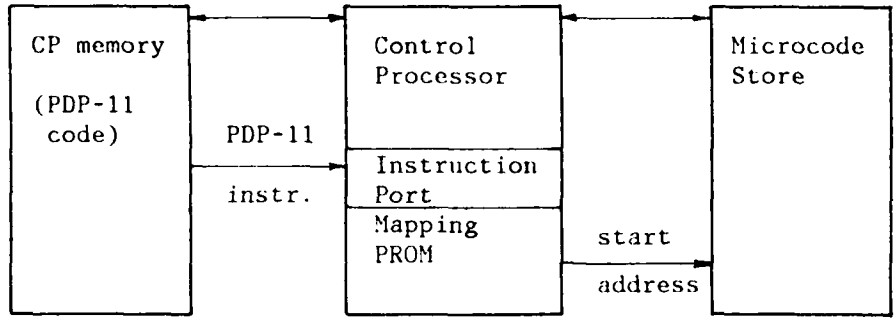


Figure 1. A schematic of the Control Processor

```
SIGNETICS BIPOLAR MICRO ASSEMBLER(A)
ADDR:    SRC    DST  MARS   C2903        C2904       CCMUX C2910  SOF         IMMOP       ERROR

       "MACRO FETCH ROUTINE"
       FETCH:  MOVS R7,,MAR  READ           "MAR=PC"
                 CJP INTREQ  TADDR=INTROUT "JUMP TO INTERRUPT HANDLER IF
                                            INTERRUPT PRESENT";
0031: 000111 00000 1     011011100 0000001001000000 010    0011  10000 0000000000011100 0
               INC2 ,R7                    "UPDATE PC, MORE";
0032: 000000 00111 0     000000100 0100001001000000 000    1110  01111 0000000000000000 0
               DST=MIR  JMAP               "LOAD MIR, JUMP TO START ADDR";
0033: 000000 10100 0     010011100 0000001001000000 000    0010  01111 0000000000000000 0

     "SINGLE OPERAND INSTRUCTIONS"
     "CLR, COM, INC, DEC, NEG, ADC, SBC, TST, ROR, ROL, ASR, SXT, MARK,
      MFPI, MTPI"
     "SINGLE OPERAND INSTRUCTIONS - CLASS SO1"
                                          " DST MODE EQ 0"
     CLR1:      CLR ,RD   MSR  NEXT       "CLR OP, END";
0374: 000000 11000 0     100010100 0000001001001010 000    0011  01111 0000000000011001 0
     COM1:      NOT_D ,RD   MSRC1  NEXT    "COM OP, END";
0375: 000000 11000 0     010110100 0000000110001010 000    0011  01111 0000000000011001 0
     INC@1:     INC1 ,RD   MSRNOC  NEXT    "INC OP, END";
0376: 000000 11000 0     000000100 0000001001000010 000    0011  01111 0000000000011001 0
     DEC1:      D_SUB_S ONE,RD  MSRNOC  NEXT "DEC OP, END";
0377: 001100 11000 0     000110100 0100001001000010 000    0011  01111 0000000000011001 0
     NEG1:      NOT_D ,RD  111_12=1  MSRC1  NEXT "NEG OP, END";
0400: 000000 11000 0     010110100 0100000110001010 000    0011  01111 0000000000011001 0
     ADC1:      MOVD ,RD  111_12 3  MSR NEXT "ADC OP, END";
0401: 000000 11000 0     010010100 1100001001001010 000    0011  01111 0000000000011001 0
     SBC1:      MOVS ZERO,TEMP  111_12 3  IO_5=MC "TEMP CIN, MORE";
0402: 001011 01111 0     011010100 1100001001000000 000    1110  01111 0000000000000000 0
               D_SUB_S TEMP,RD  MSRC1  NEXT "DST-TEMP, END";
0403: 001111 11000 0     000110100 0100000110001010 000    0011  01111 0000000000011001 0
     TST1:      MOVS RD  MSR  NEXT          "TST OP, END";
0404: 011000 00000 0     011011100 0000001001001010 000    0011  01111 0000000000011001 0
```

Figure 2. A sample of micro-assembler output

MICRO ASSEMBLER SYNTAX

```
        alu source dest mar 2904    usr        transfer    label      memory  loading        comment
        funct reg    reg q  shift   msr          type       or        control  of                    terminator
        defn defn    defn   control control                conditional       immediate
                                                             test              field
LABEL: <ALU > RSN,RDN,<MAR > <TST > <MSR     >    <JMP LABEL  >    <READ  > <PRESET=ONNH> "COMMENT"     ;
       <SPC_FN>       <QIN > <STO > <MSRCI   >    <JSR LABEL  >    <READB > <TADDR=ONNH >
                      <MARQ> <STOB> <MSRNOC >    <RETURN     >    <WRITE >
                             <BSX > <MSRZ    >    <NEXT       >    <WRITEB>
                             <ROR > <USR     >    <TRAP       >    <READC >
                             <ROL > <STO_PSW>    <TRNSFR COND>    <WRITEC>
                             <ASL >                               <HALT  >
```

Figure 3. The micro-assembler syntax

MNEMONIC DEFINITION

ALU	I1_4	SPC_FN	I5_8	RSN	SRC	RDN	DST	TRNSFR	C2910	COND	I0_5	CCMUX
SET_F	0	MUL	0	R0	0	R0	0	JZ	0	ULE	10H	1
D_SUB_S	1	SMUL	2	R1	1	R1	1	CJS	1	UGT	11H	1
S_SUB_D	2	INC1	4	R2	2	R2	2	JMAP	2	ULT	12H	1
S_ADD_D	3	INC2	4	R3	3	R3	3	CJP	3	UGE	13H	1
MOV_D	4	CONV	5	R4	4	R4	4	PUSH	4	UEQ	14H	1
NOT_D	5	MULF	6	R5	5	R5	5	JSRP	5	UNE	15H	1
MOV_S	6	NMZS	8	R6	6	R6	6	CJV	6	UVS	16H	1
NOT_S	7	NMZD	0AH	SP	6	SP	6	JRP	7	UVC	17H	1
CLR	8	SDIV	0CH	R7	7	R7	7	RFCT	8	UCS	1AH	1
NS_AND_D	9	FDIV	0EH	PC	7	PC	7	RPCT	9	UCC	1BH	1
S_NXOR_D	0AH			R8	8	R8	8	CRTN	0AH	ULOS	1CH	1
S_XOR_D	0BH			R9	9	R9	9	CJPP	0BH	UHI	1DH	1
S_AND_D	0CH			R10	0AH	R10	0AH	LDCT	0CH	UMI	1EH	1
S_NOR_D	0DH			ZERO	0BH	-	-	LOOP	0DH	UPL	1FH	1
S_NAND_D	0EH			ONE	0CH	-	-	CONT	0EH	MLE	20H	1
S_OR_D	0FH			TWO	0DH	-	-	TWB	0FH	MGT	21H	1
				TEMP@	0EH	TEMP@	0EH			MLT	22H	1
				TEMP	0FH	TEMP	0FH			MGE	23H	1
				RS	10H	RD	18H			MEQ	24H	1
				RD	18H	RS	1CH			MNE	25H	1
				DR0	20H	DR0	10H			MVS	26H	1
				DR1	21H	DR1	11H			MVC	27H	1
				DR2	22H	DR2	12H			MCS	2AH	1
				IMM	23H	IMM	13H			MCC	2BH	1
		FLDSEL(OFFSET,WIDTH)			30H	MIR	14H			MLOS	2CH	1
				PSW	--	QDST	-			MHI	2DH	1
										MMI	2EH	1
										MPL	2FH	1
										INTREQ		3

Figure 4. Microcode mnemonic definition

DISTRIBUTION

	Copy No.
EXTERNAL	
In United Kingdom	
The Institution of Electrical Engineers, Hitchin Herts, S65 IRJ	1
British Library, Lending Division	2
Defence Science Representative, London	Title page
In United States of America	
Cambridge Scientific Abstracts, Riverdale MD 20840	3
Counsellor, Defence Science, Washington	Title page
In Australia	
Department of Defence	
Chief Defence Scientist	4
Deputy Chief Defence Scientist	
Controller, Projects and Analytical Studies	
Superintendent, Science and Technology Programmes	
Director, Joint Intelligence Organisation (DSTI)	5
Document Exchange Centre Defence Information Services Branch for:	
Microfilming	6
United Kingdom, Defence Research Information Centre (DRIC)	7
United States, Defense Technical Information Center	8 - 19
Canadian, Director, Scientific Information Services	20
New Zealand, Ministry of Defence	21
National Library of Australia	22
Director General, Army Development (NSO), Russell Offices for ABCA STandardisation Officers	
UK ABCA representative, Canberra	23
US ABCA representative, Canberra	24
Canada ABCA representative, Canberra	25
NZ ABCA representative, Canberra	26

Defence Library, Campbell Park	27
Library, Aeronautical Research Laboratories	28
Library, Materials Research Laboratories	29
Library, H Block, Victoria Barracks	30
Library, RAN Research Laboratory	31
Department of Defence Support	
Deputy Secretary A	32
Deputy Secretary B	
Controller, Aircraft Guided Weapons and Electronic Supply	
Controller, Munitions Supply	
Library, DDS Central Office	33
Director, Industry Development, Adelaide	Title page
WITHIN DRCS	
Director, Electronics Research Laboratory	34
Superintendent, Radar Division	35
Principal Engineer, Workshops Branch	36
Principal Officer, Cybernetic Electronics Group	37
Principal Officer, Radio Group	38
Principal Officer, Jindalee Development Group	39
Principal Officer, Microwave Radar Group	40
Principal Officer, Jindalee Project Group	41
Principal Officer, Systems Integration Group	42
Principal Officer, Digital Systems Engineering Group	43
Principal Officer, Computer Aided Processes Group	44
Mr P.C. Drewer, Cybernetic Electronics Group	45
Mr J. Ziukelis, Cybernetic Electronics Group	46
Mr R. Schenk, Cybernetic Electronics Group	47
Mr M.J. Denison, Jindalee Project Group	48
Mr G. Brimble, Cybernetic Electronics Group	49

Author	50
DRCS Library	51 - 52
Spares	53 - 63

DOCUMENT CONTROL DATA SHEET

Security classification of this page: UNCLASSIFIED

1 DOCUMENT NUMBERS	2 SECURITY CLASSIFICATION
AR Number: AR-002-802	a. Complete Document: Unclassified
Series Number: ERL-0286-TM	b. Title in Isolation: Unclassified
Other Numbers:	c. Summary in Isolation: Unclassified

3 TITLE

THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO ARRAY PROCESSOR

4 PERSONAL AUTHOR(S):

D.J. Heilbronn

5 DOCUMENT DATE:

August 1983

6 6.1 TOTAL NUMBER OF PAGES 51

6.2 NUMBER OF REFERENCES: 9

7 7.1 CORPORATE AUTHOR(S):

Electronics Research Laboratory

7.2 DOCUMENT SERIES AND NUMBER
Electronics Research Laboratory
0286-TM

8 REFERENCE NUMBERS

a. Task: DEF 77/036

b. Sponsoring Agency: DEPT. OF DEFENCE

9 COST CODE:

10 IMPRINT (Publishing organisation)

Defence Research Centre Salisbury

11 COMPUTER PROGRAM(S) (Title(s) and language(s))

12 RELEASE LIMITATIONS (of the document):

Approved for Public Release

Security classification of this page: UNCLASSIFIED

Security classification of this page: UNCLASSIFIED

13 ANNOUNCEMENT LIMITATIONS (of the information on these pages):

No limitation.

14 DESCRIPTORS:

a. EJC Thesaurus Terms

Computer programming
Microprogramming
Signal processing
Radar signals

b. Non-Thesaurus Terms

Array oriented processor
PDP-11 assembler language

15 COSATI CODES:

09020

16 SUMMARY OR ABSTRACT:
(if this is security classified, the announcement of this report will be similarly classified)

A high speed array processor (ARO) has been designed for the processing of radar data in real time. The operation of the ARO is supervised by a bit-slice microprocessor (the Control Processor). This document contains a description of the microcode which was written for the Control Processor to enable it to interpret PDP-11 assembler code.

The official documents produced by the Laboratories of the Defence Research Centre Salisbury are issued in one of five categories: Reports, Technical Reports, Technical Memoranda, Manuals and Specifications. The purpose of the latter two categories is self-evident, with the other three categories being used for the following purposes:

Reports	:	documents prepared for managerial purposes.
Technical Reports	:	records of scientific and technical work of a permanent value intended for other scientists and technologists working in the field.
Technical Memoranda	:	intended primarily for disseminating information within the DSTO. They are usually tentative in nature and reflect the personal views of the author.